FCA&A 2015

Proceedings of the International
Workshop on Formal Concept Analysis and Applications
(co-located with ICFCA 2015)

Universidad de Málaga (Dept. Matemática Aplicada), Spain

ISBN 978-84-606-7410-8

The 13th Intl Conf on Formal Concept Analysis

Formal Concept Analysis and Applications

FCA&A 2015

Nerja (Málaga), Spain
June 23–26, 2015

(an ICFCA workshop)

Edited by

Manuel Ojeda-Aciego
Jaume Baixeries
Christian Sacarea

Proceedings of FCA&A 2015 (co-located with ICFCA 2015)
© Manuel Ojeda-Aciego, Jaume Baixeries, Christian Sacarea, Editors

This work is subject to copyright. All rights reserved. Reproduction or publication of this material, even partial, is allowed only with the authors' and editors' permission.

Technical Editor:
Manuel Ojeda-Aciego, aciego@uma.es

Table of Contents

Preface

Formal Concept Analysis (FCA) is a mathematical field rooted in lattice and order theory which, although being of such a theoretical nature, has proved to be of interest to various applied fields such as Knowledge Discovery and Data Mining, Database Theory, Data Visualization, and many others.
The 13th International Conference on Formal Concept Analysis (ICFCA) took place from the 23th to the 26th of June, 2015 in Nerja (Málaga), Spain, organized by members of the Universidad de Málaga. The International Workshop on Formal Concept Analysis and Applications (FCA&A) was organized in conjunction to ICFCA, and consisted of the seven works included in this volume.
Our deepest gratitude goes to all the authors of submitted papers, the Program Committee members, and the external reviewers. Working with the efficient and capable team of local organizers was a constant pleasure. We are deeply indebted to all of them for making this conference a successful forum on FCA.
Last, but not least, we are most grateful to the organizations that sponsored this event: the Universidad de Málaga, Andalucía Tech (International Campus of Excellence), the Patronato de Turismo de la Costa del Sol and the Área de Turismo del Ayuntamiento de Nerja, all in Spain. Finally, we would like to emphasize the great help of EasyChair for making the technical duties easier.

June 2015

Manuel Ojeda-Aciego
Jaume Baixeries
Christian Sacarea

Depth-first Search for Pseudo-intents through Minimal Transversals

Alexandre Bazin

contact@alexandrebazin.com

Abstract. The enumeration of pseudo-intents is a long-standing problem in which the order plays a major role. In this paper, we present new algorithms that enumerate pseudo-intents in orders that do not necessarily respect the inclusion relation. We show that, confirming established results, this enumeration is equivalent to computing minimal transversals in hypergraphs a bounded number of times.

1 Introduction

Formal concept analysis, as a field of applied mathematics, is interested in the patterns that can be found in data taking the form of a set of objects described by attributes. Among these patterns are the *implications*, relations between attribute sets A and B representing the fact that every object described by A is also described by B. As often with this type of relation, interest revolves around a small, non redundant set of implications. Such a set, the *Duquenne-Guigues* basis, is constructed using the notion of *pseudo-intent*. Indeed, computing the Duquenne-Guigues basis, and thus all the implications in a data set, is equivalent to enumerating all the pseudo-intents. In [15], it is shown that the number of pseudo-intents can be exponential in the number of attributes. It has been proven in [6, 2] that pseudo-intents cannot be enumerated with a polynomial delay in lectic or reverse lectic order. However, to the best of our knowledge, no such result exists for other orders. The best-known algorithm, Next Closure [11], enumerates only in the lectic order. In a previous work [4], we proposed an exponential delay algorithm to enumerate in any order that extends the inclusion order. In order to continue the study of the influence of the order on the complexity of this enumeration problem, we propose here two new algorithms that, together, can enumerate pseudo-intents without respecting the inclusion.

Our algorithms enumerate pseudo-intents incrementally, i.e. given a formal context and a set of previously found implications, they return a new pseudo-intent. In order to do this, we define new conditions under which a pseudo-intent P can be recognized using the lower covers of P in the lattice of attribute sets closed under the implications already found. This allows us to build algorithms that, instead of starting from $\emptyset$ and adding attributes to construct sets, start from $\top$ and remove attributes, resulting in a depth-first search in the lattice. We show that both constructing and recognizing a pseudo-intent can be done

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors. Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

by computing a bounded number of times the minimal transversals of some hypergraph, which makes it easy to bound the delay and isolate special, easier cases. This result establishes a link with the proof in [6] that enumerating pseudo-intents without order is at least as hard as computing minimal transversals.

We start by recalling in Section 2 the basic definitions of FCA and results on the enumeration of pseudo-intents and minimal transversals. In Section 3, we present the definition of a recognizable pseudo-intent that we use along with the properties needed for our algorithms. Finally, Sections 4 and 5 are dedicated to the two algorithms, an example and a brief study of their delay.

2 Preliminaries

2.1 Basics

In formal concept analysis, data takes the form of a *formal context*. A formal context is a triple $\mathcal{C} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$ in which $\mathcal{O}$ is a set of objects, $\mathcal{A}$ is a set of attributes and $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ a relation that maps attributes to objects. An object $o \in \mathcal{O}$ is said to be *described* by an attribute $a \in \mathcal{A}$ if $(o, a) \in \mathcal{R}$. Together with the context, there are two $\cdot'$ operators defined as

$$\cdot' : 2^{\mathcal{O}} \mapsto 2^{\mathcal{A}}$$

$$O' = \{a \in \mathcal{A} \mid \forall o \in O,\ (o, a) \in \mathcal{R}\}$$

$$\cdot' : 2^{\mathcal{A}} \mapsto 2^{\mathcal{O}}$$

$$A' = \{o \in \mathcal{O} \mid \forall a \in A,\ (o, a) \in \mathcal{R}\}$$

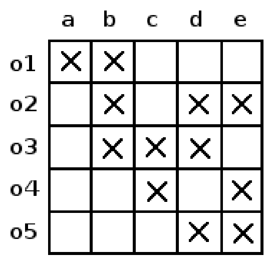
Their composition $\cdot''$ is a closure operator. A set of attributes $A = A''$ closed under $\cdot''$ is called an *intent*.

2.2 Implications

An *implication* is a relation between two attribute sets A and B, noted $A \rightarrow B$. It is said to *hold* in the context if and only if $A' \subseteq B'$ or, in other words, if and only if every object described by A is also described by B. A set $\mathcal{I}$ of implications is a *basis* if and only if every implication that holds in the context can be derived from $\mathcal{I}$ using Armstrong's rules :

$$\frac{B \subseteq A}{A \rightarrow B}, \quad \frac{A \rightarrow B}{A \cup C \rightarrow B \cup C}, \quad \frac{A \rightarrow B, B \rightarrow C}{A \rightarrow C}$$

A *pseudo-intent* (or *pseudo-closed set*) P is a set of attributes that contains the closure of all its subsets and is not an intent. The set $\mathcal{B} = \{P \rightarrow P'' \mid P$



	a	b	c	d	e
o1	×	×			
o2		×		×	×
o3		×	×	×	
o4			×		×
o5				×	×

Fig. 1. A formal context

is a pseudo-intent} is the implication basis with the minimum cardinality and is called the *Duquenne-Guigues basis* [12]. As such, computing the Duquenne-Guigues basis, and thus all the implications, is enumerating all the pseudo-intents. As one of the great problems in FCA, it has been abundantly studied [12, 15, 6, 17, 4, 3, 16, 1, 19]. As the number of pseudo-intents can be exponential in the size of the context, preventing a polynomial algorithm, the attention is instead focused on the delay, i.e. the time between two pseudo-intents and the last pseudo-intent and the end of the algorithm. It has been shown that pseudo-intents cannot be enumerated with a polynomial delay in lectic [6] or reverse lectic order [2]. For the enumeration without order, it is shown in [6] that it is at least as hard as computing the minimal transversals of a hypergraph. However, the upper bound is not yet known. The problem of recognizing a pseudo-intent has been studied in [1] and found to be coNP-complete.

2.3 Logical Closures

A set of implications $\mathcal{I}$ gives rises to a *logical closure* operator $\mathcal{I}(\cdot)$ defined as

$$A^{+} = A \cup \{C \mid B \rightarrow C \in \mathcal{I} \text{ and } B \subseteq A\}$$
$$\mathcal{I}(A) = A^{++\cdots+} \text{ (up to saturation)}$$

Similarly, we have the *logical pseudo-closure* operator $\mathcal{I}^{-}(\cdot)$ defined as

$$A^{\circ} = A \cup \{C \mid B \rightarrow C \in \mathcal{I} \text{ and } B \subset A\}$$
$$\mathcal{I}^{-}(A) = A^{\circ\circ\cdots\circ} \text{ (up to saturation)}$$

It is known that intents are closed under both $\mathcal{B}^{-}(\cdot)$ and $\mathcal{B}(\cdot)$, whereas a pseudo-intent P is closed under $\mathcal{B}^{-}(\cdot)$ and $\mathcal{I}(\cdot)$ for $\mathcal{I} \subseteq \mathcal{B} \setminus \{P \rightarrow P''\}$.

An attribute set Y is said to be a *generator* of an attribute set X under $\mathcal{I}(\cdot)$ (resp. $\mathcal{I}^{-}(\cdot)$) if $\mathcal{I}(Y) = X$ (resp. $\mathcal{I}^{-}(Y) = X$). It is a *minimal generator*

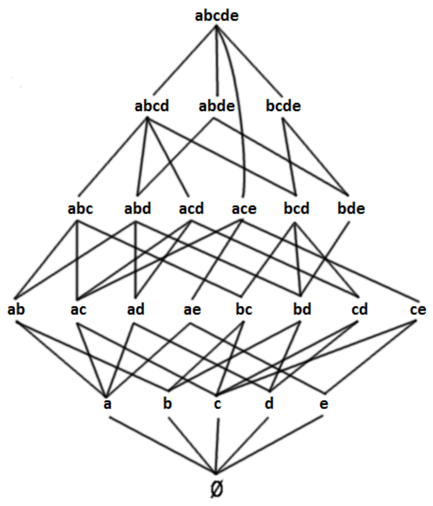


Fig. 2. $\Phi_{\mathcal{I}}$ with $\mathcal{I} = \{de \rightarrow bde, be \rightarrow bde\}$

if there is no generator Z of X such that $Z \subset Y$. We will use $Gen_{\mathcal{I}}(X)$ to denote the set of all minimal generators of a set X under $\mathcal{I}(\cdot)$. The complexity of computing the minimal generators of a set has been studied in [14]. The number of minimal generators can be exponential in the number of attributes and they can be enumerated with an incremental polynomial delay.

We shall use $\Phi_{\mathcal{I}}$ to denote the lattice of attribute sets closed under $\mathcal{I}(\cdot)$ ordered by the inclusion relation. For an attribute set A, the set of pseudo-intents P in $\mathcal{I}$ such that $P \subseteq A$ and $P'' \not\subseteq A$ will be denoted by $\mathcal{P}_{\mathcal{I}}(A)$.

2.4 Minimal Transversals

The problem of computing *minimal transversals* in hypergraphs or, equivalently, the prime conjunctive normal form of the dual of a Boolean function (otherwise called *monotone dualization*), have been largely studied in the literature [13, 10, 8, 9, 7].

Given a hypergraph $\mathcal{H} = (E, \mathcal{V})$ where E is a set of edges and $\mathcal{V} \subseteq 2^E$ is a set of vertices, a *transversal* $T \subseteq E$ is a set of edges such that $\forall V \in \mathcal{V}$, $T \cap V \neq \emptyset$. A transversal is *minimal* if none of its subsets is a transversal. We shall use $\mathcal{T}r(V)$ to denote the set of minimal transversals of a set of vertices V. For example, $\mathcal{T}r(\{ace, bce, cde\}) = \{c, e, abd\}$.

The problem of computing all the minimal transversals of a hypergraph can be solved in quasi-polynomial total time [9, 10, 13]. It is currently not known whether it is possible to enumerate the transversals with a polynomial delay in the general case. However, many special cases are known to have an output-polynomial delay. Their references can be found in [9].

In this work, we cannot presuppose the nature of the hypergraph and so we are interested in the general case. The problem still being actively studied, we will suppose that we have a blackbox algorithm nextTrans that enumerates the minimal transversals of a set of vertices X. We suppose, for ease of writing, that the transversals T_i this algorithm enumerates in some arbitrary order $T_1 < T_2 < ... < T_n$ with $\text{nextTrans}(T_i, X) = T_{i+1}$ and $\text{nextTrans}(T_n, X) = \emptyset$. For example, Berge Multiplication [5] can be adapted to take the role of nextTrans. It is important to note that a first transversal can be computed in polynomial time by simply removing attributes until the set stops being a transversal.

3 Recognizing a Pseudo-Intent

We know that an attribute set is a pseudo-intent if and only if it contains the closure of all its subsets that are pseudo-intents. As for the problem of recognizing pseudo-intents, two cases have been studied. When only a context and a set A are provided, checking whether A is a pseudo-intent is coNP-complete [1]. An algorithm has been proposed in [19] that runs in $O(2^{|\mathcal{A}|})$. When the context and A are provided alongside all the pseudo-intents (and thus implications) contained in A, checking whether A is a pseudo-intent is easier as we only have to verify the closedness of A for the logical closure and $\cdot''$, which implies a runtime polynomial in the size of the implication set. It is this method that is used in algorithms such as Next Closure or the one we proposed in [4] and its drawback is that it forces an enumeration in an order that extends the inclusion order.

In this paper, we are interested in enumerating pseudo-intents in orders that do not extend the inclusion and, as such, we cannot suppose that we know the closure of all the subsets of a set before attempting to recognize its pseudo-closedness. Hence, we propose to consider the problem of recognizing a pseudo-intent A given a context and a set of subsets of A that is not necessarily complete.

Proposition 1. *An attribute set $A \in \Phi_{\mathcal{I}}$ is a pseudo-intent if $A \neq A''$ and all its lower covers in $\Phi_{\mathcal{I}}$ are closed.*

Proof. Let us suppose that A is not closed and that all its lower covers are closed. Let B be a lower cover of A and X be a subset of A. If $B \subset X$, then $\mathcal{I}(X) = A$ and $X'' = A''$. If $X \subseteq B$ and $X'' \not\subseteq B$, then B cannot be closed and we have a contradiction. Therefore, $X'' \subseteq B$. Since the closure of every subset of A is either A'' or contained in A, the set A is a pseudo-intent. □

This proposition states that some pseudo-intents can be recognized in $\Phi_{\mathcal{I}}$ only by looking at their lower covers. As such, when $\mathcal{I}$ is a subset of the Duquenne-Guigues basis, a new pseudo-intent can be found by looking in $\Phi_{\mathcal{I}}$ for sets that only have intents as lower covers.

Definition 1. *A pseudo-intent that can be recognized using Proposition 1 with a set of implication $\mathcal{I}$ is said to be recognizable. The set of all the implications that are recognizable with $\mathcal{I}$ is denoted by $Rec(\mathcal{I})$.*

Proposition 2. $\forall \mathcal{I} \subset \mathcal{B}, Rec(\mathcal{I}) \neq \emptyset$

Proof. Let P be minimal among pseudo-intents that are not premises of implications in $\mathcal{I}$. If there is a lower cover X of P in $\Phi_{\mathcal{I}}$ that is not closed, then there is a set $A \subset X$ in $\Phi_{\mathcal{I}}$ such that $A \rightarrow A''$ holds in the context. If $A'' \subseteq P$, then P is not minimal. If $A'' \nsubseteq P$, then P is not a pseudo-intent. Both cases lead to contradictions so all the lower covers of P are closed and P is recognizable from $\mathcal{I}$. Since we have $\mathcal{I} \subset \mathcal{B}$, the set of pseudo-intents that are not a premise of $\mathcal{I}$ is not empty, so it has minimal elements. Hence, the set of pseudo-intents that are recognizable from $\mathcal{I}$ is not empty if $\mathcal{I} \subset \mathcal{B}$. $\square$

We have that, even though some unknown pseudo-intents may not be recognizable, there are always additional recognizable pseudo-intents for any subset of the Duquenne-Guigues basis. This ensures that we are able to enumerate all the pseudo-intents with an algorithm that finds recognizable ones.

Proposition 3. *Let A and B be two elements of $\Phi_{\mathcal{I}}$. The set B is a lower cover of A if and only if $B = A \setminus C$ with C a minimal transversal of $Gen_{\mathcal{I}}(A)$.*

Proof. Let C be minimal such that $\forall G \in Gen_{\mathcal{I}}(A)$, $G \cap C \neq \emptyset$. For any attribute $i \in A \setminus C$, the set $(A \setminus C) \cup \{i\} = A \setminus (C \setminus \{i\})$ contains a minimal generator of A because of the minimality of C. Therefore, any B between $A \setminus C$ and A is such that $\mathcal{I}(B) = A$ and cannot be in $\Phi_{\mathcal{I}}$. Hence, $A \setminus C$ is a lower cover of A.

Let B be a lower cover of A in $\Phi_{\mathcal{I}}$. By definition, $\mathcal{I}(B \cup \{i\}) = A$ for any attribute $i \in A \setminus B$ so there is a subset C of A such that $i \in C$ and $(C \setminus \{i\}) \subseteq B$ that is a minimal generator of A. $\square$

This proposition states that the lower covers of a set can be computed from and depend on its minimal generators. As such, the number of lower covers can be exponential in the number of minimal generators which can itself be exponential in the size of the attribute set [14].

Definition 2. *For any two attribute sets A and B, we say that B is reachable from A if and only if $B \subset A$ and there is an ordering $x_1 < x_2 < ... < x_n$ of the elements of $A \setminus B$ such that, for every $m < n$, $A \setminus \{x_1, x_2, ..., x_m\}$ is not closed under $\mathcal{I}$.*

In other words, B is reachable from A if you can obtain B by removing attributes from A and only go through sets that are not closed under $\mathcal{I}$. Evidently, minimal reachable sets are elements of $\Phi_{\mathcal{I}}$.

Proposition 4. *An attribute set $A \in \Phi_{\mathcal{I}}$ is a minimal recognizable pseudo-intent if and only if A is not an intent and all the minimal attributes sets reachable from A are intents.*

Algorithm 1 *reachableSets*(A)

```
Require: An attribute set $A$, an implication set $\mathcal{I}$
 1: $R = \emptyset$
 2: for every $i \in A$ do
 3:     $G = \bigcup_{P \in \mathcal{P}_{\mathcal{I}}(A \setminus \{i\})} Gen_{\mathcal{I}}(P)$
 4:     $C$ = The first transversal of $G$
 5:     while $C \neq \emptyset$ do
 6:         $B = A \setminus (\{i\} \cup C)$
 7:         if $\mathcal{P}_{\mathcal{I}}(B) \neq \emptyset$ then
 8:             $R = R \cup reachableSets(B)$
 9:         else
10:             $R = R \cup \{B\}$
11:         end if
12:         $C = nextTrans(C, G)$
13:     end while
14: end for
15: Return $R$
```

Proof. If A is a minimal recognizable pseudo-intent, then every subset of A in $\Phi_{\mathcal{I}}$ is an intent. Thus every minimal reachable set is an intent.

Let us suppose that every minimal reachable subset of A is an intent and $A \neq A''$. For any set X reachable from A and any attribute set $P \subseteq X$, if $P'' \not\subseteq A$, then X cannot be an intent. Every minimal reachable subset of A being an intent, A contains the closure of all its subsets so it is a pseudo-intent. If P is a pseudo-intent that is not in $\mathcal{I}$, subsets of $A \setminus \{i\}$ with $i \in P'' \setminus P$ cannot be intents so either P or a superset of P in $\Phi_{\mathcal{I}}$ that is not an intent is reachable from A. Hence, A is a minimal recognizable pseudo-intent. $\square$

A minimal recognizable pseudo-intent can therefore be recognized by computing the sets of its minimal reachable subsets. These minimal reachable sets can be computed using Algorithm 1. By definition, reachable sets can be computed by removing from A attributes that play a role in the logical closure, i.e. attributes in minimal generators of pseudo-intents $P \subset A$ such that $P'' \not\subseteq A \setminus X$ for some $A \setminus X$ reachable from A. The minimal transversals T make up all the possible ways to remove minimal generators of pseudo-intents and, thus, all the sets T such that for every $Y \subset T$, $A \setminus Y$ cannot be logically closed. As such, removing minimal transversals of the generators of pseudo-intents used in the logical closure of a set produces a reachable set. The set $A \setminus T$ is not always logically closed so the algorithm is recursively applied until an element of $\Phi_{\mathcal{I}}$ is reached.

The algorithm is not optimal as some reachable sets can be computed multiple times and the sets reachable from a set A that is not an element of $\Phi_{\mathcal{I}}$ could be computed directly from $\mathcal{P}_{\mathcal{I}}(A)$ instead of its direct subsets. However, it is sufficient to give us an upper bound of the runtime. For each recursive call, the algorithm computes $\mathcal{P}_{\mathcal{I}}(A \setminus \{i\})$ for every attribute $i \in A$. For each of these attributes, it then computes the minimal transversals of $\bigcup_{P \in \mathcal{P}_{\mathcal{I}}(A \setminus \{i\})} Gen_{\mathcal{I}}(P)$. There is another recursive call for each pseudo-intent found missing so the runtime is bounded by $O(|\mathcal{A}| \times |\mathcal{I}| \times T)$ where T is the complexity of computing the transversals.

In the rest of this paper, we will suppose that we have an algorithm nextReach that enumerates minimal reachable sets in the same fashion as nextTrans.

4 Computing a Recognizable Pseudo-Intent

4.1 Algorithm

We want to incrementally compute the pseudo-intents in a formal context. In the beginning, we only have the formal context $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ and an empty set of implications $\mathcal{I}$. The corresponding $\Phi_{\mathcal{I}}$ is the boolean lattice of subsets of $\mathcal{A}$. In order to find a first pseudo-intent, we can look for a recognizable set (Proposition 1) in $\Phi_{\mathcal{I}}$. We know that a non-closed set contains a pseudo-intent. Thus, we can start with $\mathcal{A}$ and recursively remove attributes as long as we obtain non-closed sets. When we cannot remove attributes without obtaining an intent, the set is a recognizable pseudo-intent and we have a first implication. This corresponds to the algorithm presented in [6] that computes a first pseudo-intent. We can then generalize the algorithm for other pseudo-intents. As $\mathcal{I}$ contains a new implication, sets disappear from $\Phi_{\mathcal{I}}$ and we now have to look in the new lattice for a second pseudo-intent. As it is not a Boolean lattice anymore, we cannot simply remove attributes to obtain lower covers so we have to use Proposition 3 to compute them. By going through the lattice using non-closed lower covers of sets until we find a set respecting Proposition 1, we can compute a second pseudo-intent. Pseudo-intents can thus be computed incrementally until $\mathcal{A}$ no longer has any non-closed lower cover. Algorithm 2, given a context, a subset $\mathcal{I}$ of the Duquenne-Guigue Basis and an attribute set $A \in \Phi_{\mathcal{I}}$, uses this method to compute a new pseudo-intent $P \subseteq A$.

Proposition 5. *nextPI($\mathcal{A}$,$\mathcal{I}$) returns either $\mathcal{A}$ or a pseudo-intent that is not in $\mathcal{I}$.*

Proof. Let X be a set in $\Phi_{\mathcal{I}}$. If $X'' \neq X$, then there is a pseudo-intent $P \subseteq X$. If a lower cover of X is not closed, then there is a pseudo-intent $X \subset S$. The algorithm returns the first set it finds that has all its lower covers closed. If this set is not $\mathcal{A}$, then it is not closed and is thus a pseudo-intent. $\square$

Algorithm 3 uses Algorithm 2 to enumerate pseudo-intents iteratively until it returns $\mathcal{A}$. It is sound but not complete, as exemplified below.

Algorithm 2 $nextPI(A, \mathcal{I})$

```
Require: Attribute set $A$, Implication set $\mathcal{I}$
 1: $P = A$
 2: $G = Gen_{\mathcal{I}}(A)$
 3: $C =$ The first transversal of $G$
 4: while $C \neq \emptyset$ do
 5:   $S = A \setminus C$
 6:   if $S'' \neq S$ then
 7:     $P = nextPI(S)$
 8:     $C = \emptyset$
 9:   else
10:     $C = nextTrans(C, G)$
11:   end if
12: end while
13: RETURN $P$
```

Algorithm 3 $allPI$

```
1: $\mathcal{I} = \emptyset$
2: $A = nextPI(A, \mathcal{I})$
3: while $A \neq \mathcal{A}$ do
4:   $\mathcal{I} = \mathcal{I} \cup \{A \rightarrow A''\}$
5:   $A = nextPI(A, \mathcal{I})$
6: end while
7: Return $\mathcal{I}$
```

4.2 Example

Let us consider an arbitrary context for which the Duquenne-Guigues basis is $\{de \rightarrow bde, bc \rightarrow bcd, ac \rightarrow abcd, be \rightarrow bde, bcde \rightarrow abcde\}$. We want to find its pseudo-intents using Algorithm 3 :

The set of implications is initially empty. We start with $abcde$.

It has a single minimal generator, itself. The first minimal transversal of $Gen(abcde)$ is a. The set $abcde \setminus a = bcde$ is not an intent so we continue recursively with it.

The set $bcde$ has a single minimal generator, itself. The first minimal transversal of $Gen(bcde)$ is b. The set $bcde \setminus b = cde$ is not an intent so we continue recursively with it.

The set cde has a single minimal generator, itself. The first minimal transversal of $Gen(cde)$ is c. The set $cde \setminus c = de$ is not an intent so we continue recursively with it.

The set de has a single minimal generator, itself. The first minimal transversal of $Gen(de)$ is d. The set $de \setminus d = e$ is an intent. The second and last minimal transversal is e. The set $de \setminus e = d$ is an intent. The algorithm returns de as the first pseudo-intent which gives us $\mathcal{I} = \{de \rightarrow bde\}$.

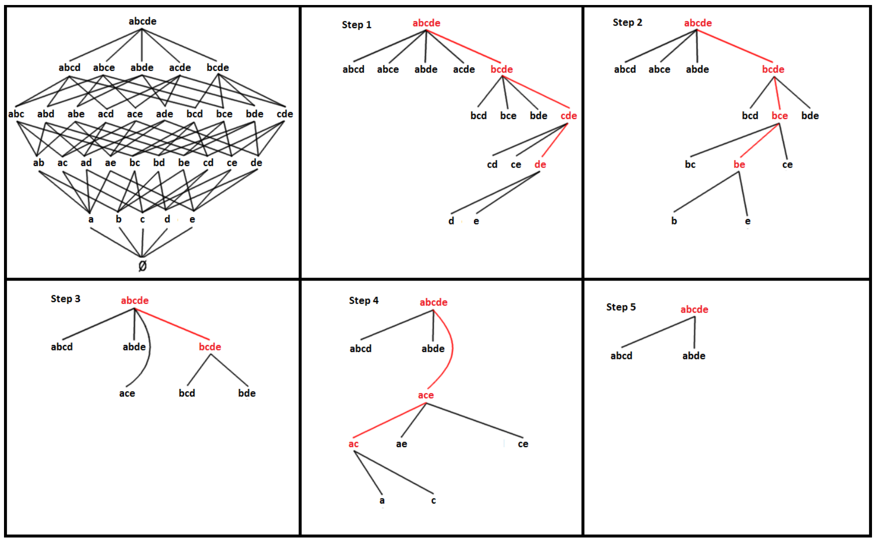


Fig. 3. The five steps of the algorithm. In red the sets considered by the recursive calls, in black the lower covers of those sets in $\Phi_{\mathcal{I}}$.

We start again with *abcde*. It has a single minimal generator, *acde*. The first minimal transversal of $Gen(abcde)$ is a. The set $abcde \setminus a = bcde$ is not an intent so we continue recursively with it.

The set *bcde* has a single minimal generator, *cde*. The first minimal transversal of $Gen(cde)$ is c. The set $bcde \setminus c = bde$ is an intent. The second minimal transversal is d. The set $bcde \setminus d = bce$ is not an intent so we continue recursively with it.

The set *bce* has a single minimal generator, itself. The first minimal transversal of $Gen(bce)$ is b. The set $bce \setminus b = ce$ is an intent. The second minimal transversal is c. The set $bce \setminus c = be$ is not an intent so we continue recursively with it.

The set *be* has a single minimal generator, itself. The first minimal transversal of $Gen(be)$ is b. The set $be \setminus b = e$ is an intent. The second and last minimal transversal is e. The set $be \setminus e = b$ is an intent. The algorithm returns *be* as the second pseudo-intent which gives us $\mathcal{I} = \{de \rightarrow bde, be \rightarrow bde\}$.

We start again with *abcde*. It has two minimal generators, *acde* and *abce*. The first minimal transversal of $Gen(abcde)$ is a. The set $abcde \setminus a = bcde$ is not an intent so we continue recursively with it.

The set *bcde* has two minimal generators, *bce* and *cde*. The first minimal transversal of $Gen(cde)$ is c. The set $bcde \setminus c = bde$ is an intent. The second minimal transversal is bd. The set $bcde \setminus bd = ce$ is an intent. The third and last minimal transversal is e. The set $bcde \setminus e = bcd$ is an intent. The algorithm returns $bcde$ as the third pseudo-intent which gives us $\mathcal{I} = \{de \rightarrow bde, be \rightarrow bde, bcde \rightarrow abcde\}$.

We start again with $abcde$. It has two minimal generators, cde and bce. The first minimal transversal of $Gen(abcde)$ is bd. The set $abcde \setminus bd = ace$ is not an intent so we continue recursively with it.

The set ace has a single minimal generator, itself. The first minimal transversal of $Gen(ace)$ is a. The set $ace \setminus a = ce$ is an intent. The second minimal transversal is c. The set $ace \setminus c = ae$ is an intent. The third minimal transversal is e. The set $ace \setminus e = ac$ is not an intent so we continue recursively with it.

The set ac has a single minimal generator, itself. The first minimal transversal of $Gen(ace)$ is a. The set $ac \setminus a = c$ is an intent. The second and last minimal transversal is c. The set $ac \setminus c = a$ is intent. The algorithm returns ac as the fourth pseudo-intent which gives us $\mathcal{I} = \{de \rightarrow bde, be \rightarrow bde, bcde \rightarrow abcde, ac \rightarrow abcd\}$.

We start again with $abcde$. It has three minimal generators, ace, cde and bce. The first minimal transversal of $Gen(abcde)$ is c. The set $abcde \setminus c = abde$ is an intent. The second minimal transversal is e. The set $abcde \setminus e = abcd$ is an intent. The third and last minimal transversal is abd. The set $abcde \setminus abd = ce$ is an intent. The algorithm ends.

The algorithm has found four pseudo-intents but is unable to find the set bc when considering the minimal transversals, and thus the sets, **in that particular order**. Indeed, knowing $be \rightarrow bde$ and $de \rightarrow bde$ makes $bcde$ recognizable and blocks every path from $abcde$ to bc in $\Phi_{\mathcal{I}}$. Finding bc before either de or be solves the problem.

4.3 Complexity

The nature of Algorithm 2 makes it easy to bound the delay between finding two pseudo-intents. As the algorithm starts from $\mathcal{A}$ and removes attributes until it ends, it performs a maximum of $|\mathcal{A}|$ recursive calls before finding a pseudo-intent. In a call, three different tasks are performed : computing the closure of a set, computing the set of minimal generators and computing the set of all lower covers. The closure of a set is known to be computable in polynomial time. Computing $Gen_{\mathcal{I}}(A)$ is done in time polynomial in the size of the output. The computation of all the lower covers of A, as we have seen in Section 2.4, can be performed in quasi-polynomial total time, i.e. in time quasi-polynomial in the size of $n = |Gen_{\mathcal{I}}(A)|$ and $m = |\mathcal{T}r(Gen_{\mathcal{I}}(A))|$ (note that the size of m itself is bounded by $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ [18]). Hence, the delay is in $O(|\mathcal{A}| \times (C + G + T))$ with C the complexity of computing a closure, G the complexity of computing minimal generators and T the complexity of computing the lower covers.

Algorithm 4 $nextMinPI(A, \mathcal{I})$

```
Require: Attribute set $A$, Implication set $\mathcal{I}$
 1: $P = A$
 2: $C =$The first minimal set reachable from $A$
 3: while $C \neq \emptyset$ do
 4:     if $C'' \neq C$ then
 5:         $P = nextMinPI(C)$
 6:         $C = \emptyset$
 7:     else
 8:         $C = nextReach(C, A)$
 9:     end if
10: end while
11: RETURN $P$
```

An interesting point of detail is that the algorithm does not necessarily enumerate all the intents. When all the upper covers of an intent A in $\Phi_{\mathcal{I}}$ are intents themselves, A cannot be reached by the algorithm anymore. In particular, if $A \cup \{i\} = (A \cup \{i\})''$ for every $i \in \mathcal{A} \setminus A$, we are sure that A will never be considered. In the previous example (see Figure 3), we see that the intents *abd*, *ab*, *ad*, *bd*, *de* and $\emptyset$ are never considered because they do not appear as lower covers of sets. We suppose that this property helps reduce the runtime on dense contexts with a high number of intents.

5 Computing a Minimal Recognizable Pseudo-Intent

5.1 Algorithm

The problem with Algorithm 3 is that there are some pseudo-intents it cannot compute. Among those leftover pseudo-intents, there is necessarily at least one that is minimal. We have shown in Proposition 4 that a minimal recognizable pseudo-intent P can be recognized through the minimal sets reachable from P. Those reachable sets can themselves be obtained by computing the transversals of the sets of minimal generators of the pseudo-intents involved in the logical closure of the different $P \setminus \{i\}$ (Algorithm 1). Algorithm 4 uses these properties to compute a minimal recognizable pseudo-intent. In a manner similar to that of Algorithm 2, the algorithm starts with $\mathcal{A}$ and enumerates the minimal reachable sets. Once one that is not an intent is found, the algorithm is recursively applied. The algorithm ends when it finds a set with only intents as minimal reachable sets. The main difference between Algorithm 2 and Algorithm 4 is that the former performs a depth-first search in the lattice $\Phi_{\mathcal{I}}$ while the latter performs it in the directed graph in which the nodes are the elements of $\Phi_{\mathcal{I}}$ and the edges are the pairs in the "reachable" relation.

Proposition 6. *Algorithm 4 returns either $\mathcal{A}$ or a minimal pseudo-intent that is not in $\mathcal{I}$.*

Algorithm 5 *allPI*2

```
1: $\mathcal{I} = \emptyset$
2: $A = nextminPI(A, \mathcal{I})$
3: while $A \neq \mathcal{A}$ do
4:     $\mathcal{I} = \mathcal{I} \cup \{A \rightarrow A''\}$
5:     $A = nextMinPI(A, \mathcal{I})$
6: end while
7: Return $\mathcal{I}$
```

Proof. From Proposition 4 we know that if every minimal set $T \in \Phi_{\mathcal{I}}$ reachable from $A \in \Phi_{\mathcal{I}}$ is closed and A is not, then A is a minimal recognizable pseudo-intent. If $T \in \Phi_{\mathcal{I}}$ is not closed, then T contains a minimal recognizable pseudo-intent so the algorithm can be recursively called on T. The algorithm stops when it encounters a set A with only intents as minimal reachable sets. The set $\mathcal{A}$ being the only intent on which the algorithm is called, it returns either $\mathcal{A}$ or a minimal recognizable pseudo-intent. □

We can enumerate all the pseudo-intents with Algorithm 5 using Algorithm 4 in the same fashion as Algorithm 3.

Proposition 7. *Algorithm 5 is sound and complete.*

Proof. Soundness follows from Proposition 6. For any implication set $\mathcal{I} \subset \mathcal{B}$, pseudo-intent P not used in $\mathcal{I}$ and attribute set $A \in \Phi_{\mathcal{I}}$ such that $P \subset A$, no set X such that $P \subset X$ and $P'' \not\subseteq X$ can be an intent so either P or one of its non-closed supersets is reachable from A. Hence, the algorithm does not stop until every pseudo-intent has been found. □

5.2 Example

Contrary to Algorithm 3, Algorithm 5 is complete. We can either use it to compute all the pseudo-intents or combine it with Algorithm 3. Let us suppose we have used Algorithm 3, as exemplified in Section 4, and that we know the set of implications $\mathcal{I} = \{de \rightarrow bde, be \rightarrow bde, bcde \rightarrow abcde, ac \rightarrow abcd\}$. Using Algorithm 5 from there gives us the following :

We start with $abcde$. The first minimal reachable set $abcde \setminus (a \cup bd) = ce$ is an intent. The second minimal reachable set $abcde \setminus (a \cup bc) = de$ is an intent. The third minimal reachable set $abcde \setminus (a \cup e) = bcd$ is an intent. The fourth minimal reachable set $abcde \setminus (b \cup cd) = ae$ is an intent. The fifth minimal reachable set $abcde \setminus (b \cup e) = cd$ is an intent. The sixth minimal reachable set $abcde \setminus (b \cup ce) = ad$ is an intent. The seventh minimal reachable set $abcde \setminus c = abde$ is an intent. The eighth minimal reachable set $abcde \setminus (d \cup ab) = ce$ is an intent. The ninth minimal reachable set $abcde \setminus (d \cup ae) = bc$ is not an intent so we continue recursively with it.

The set bc contains no pseudo-intents so its minimal reachable sets are $bc \setminus b = c$ and $bc \setminus c = b$. They are both intents so the algorithm returns bc as a new pseudo-intent which gives us $\mathcal{I} = \{de \rightarrow bde, be \rightarrow bde, bcde \rightarrow abcde, ac \rightarrow abcd, bc \rightarrow bcd\}$.

We start again with $abcde$. The eight first minimal reachable sets computed are the same as in the previous step. The ninth reachable set $abcde \setminus cde = ab$ is an intent. The tenth and last reachable set $abcde \setminus e = abcd$ is an intent. The algorithm ends.

In this example, the pseudo-intent bc has been found after $bcde$. Using both Algorithm 3 and Algorithm 5, it is thus possible to enumerate pseudo-intents in orders that do not extend the inclusion.

5.3 Complexity

Once again, the delay between two pseudo-intents with Algorithm 5 is easy to bound. Algorithm 4 performs at most $|\mathcal{A}|$ recursive calls before returning a pseudo-intent or $\mathcal{A}$. Each call requires the computation of, at most, all the reachable sets in $\Phi_{\mathcal{I}}$. We have shown that this can be done in $O(|\mathcal{A}| \times |\mathcal{I}| \times T)$ with T the complexity of computing minimal transversals of minimal generators of pseudo-intents. Thus, the worst case delay of Algorithm 5 is $|\mathcal{I}|$ times that of Algorithm 3.

A first minimal transversal can be computed in polynomial time. As such, for any attribute set A, we can compute its first reachable set in $\Phi_{\mathcal{I}}$ in time polynomial in $|\mathcal{A}|$ and $|\mathcal{I}|$. Hence, if, for every $A \in \Phi_{\mathcal{I}}$ that is not a pseudo-intent, the reachable sets in $\Phi_{\mathcal{I}}$ are ordered in such a way that the first one is not closed, Algorithm 5 can compute (but not necessarily recognize) a pseudo-intent in time polynomial in the size of the implication set. A minimal pseudo-intent P, by definition, does not contain any other pseudo-intent so the set of its reachable set is $\{P \setminus \{p\} \mid p \in P\}$ which can be computed in polynomial time. Thus, there are always orderings of reachable sets such that Algorithm 5 can compute minimal pseudo-intents with an incremental polynomial time between two of them. However, as shown in [6], minimal pseudo-intents cannot be enumerated in output polynomial time. Indeed, even though it is possible to compute a minimal pseudo-intent in time polynomial in the size of the implication set, confirming that they have all been found is exponential because non-minimal pseudo-intents and $\mathcal{A}$ have a potentially exponential number of reachable sets.

6 Conclusion

We have proposed two algorithms for computing pseudo-intents using the computation of minimal transversals as their main operation. The first can find pseudo-intents before some of their subsets that are themselves pseudo-intents but it is not complete for some enumeration orders. The second is complete but can only enumerate in orders that respect the inclusion relation. They can be combined to obtain the enumeration order of the first with the completeness of the second. We expressed their delay as a function of the complexity of computing minimal transversals. We showed that, for some orderings of attribute sets, pseudo-intents can be reached, but not recognized, in incremental polynomial time. As both reaching and recognizing pseudo-intents, in our context, are related to the problem of computing minimal transversals in hypergraphs for which many special cases are known, we believe that this work may be useful in isolating more cases for which the enumeration of pseudo-intents is easier.

On the practical side, the fact that the algorithms do not necessarily enumerate the entirety of the set of intents should significantly reduce the runtime on dense contexts. Furthermore, the depth-first strategy used in the algorithms allows for some computations to be saved and reused. Further algorithmic optimizations and empirical comparisons with other algorithms for the same problem will be the subject of future investigations.

References

1. Mikhail A. Babin and Sergei O. Kuznetsov. Recognizing Pseudo-intents is coNP-complete. In *Proceedings of the 7th International Conference on Concept Lattices and Their Applications, Sevilla, Spain, October 19-21, 2010*, pages 294–301, 2010.
2. Mikhail A. Babin and Sergei O. Kuznetsov. Computing Premises of a Minimal Cover of Functional Dependencies is Intractable. *Discrete Applied Mathematics*, 161(6):742–749, 2013.
3. Konstantin Bazhanov and Sergei A. Obiedkov. Comparing Performance of Algorithms for Generating the Duquenne-Guigues Basis. In *Proceedings of The Eighth International Conference on Concept Lattices and Their Applications, Nancy, France, October 17-20, 2011*, pages 43–57, 2011.
4. Alexandre Bazin and Jean-Gabriel Ganascia. Enumerating Pseudo-Intents in a Partial Order. In *Proceedings of the Tenth International Conference on Concept Lattices and Their Applications, La Rochelle, France, October 15-18, 2013.*, pages 45–56, 2013.
5. Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45. Elsevier, 1984.
6. Felix Distel and Baris Sertkaya. On the Complexity of Enumerating Pseudo-intents. *Discrete Applied Mathematics*, 159(6):450–466, 2011.
7. Thomas Eiter and Georg Gottlob. Identifying the Minimal Transversals of a Hypergraph and Related Problems. *SIAM J. Comput.*, 24(6):1278–1304, 1995.
8. Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM J. Comput.*, 32(2):514–537, 2003.
9. Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational Aspects of Monotone Dualization: A Brief Survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.
10. Michael L. Fredman and Leonid Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *J. Algorithms*, 21(3):618–628, 1996.
11. Bernhard Ganter. Two Basic Algorithms in Concept Analysis. In *Preprint Technische Hochschule Darmstadt*, 1984.

12. Jean-Louis Guigues and Vincent Duquenne. Familles Minimales d'Implications Informatives Résultant d'un Tableau de Données Binaires. *Mathématiques et Sciences humaines*, 95:5–18, 1986.
13. Vladimir Gurvich and Leonid Khachiyan. On Generating the Irredundant Conjunctive and Disjunctive Normal Forms of Monotone Boolean Functions. *Discrete Applied Mathematics*, 96-97:363–373, 1999.
14. Miki Hermann and Baris Sertkaya. On the Complexity of Computing Generators of Closed Sets. In *Formal Concept Analysis, 6th International Conference, ICFCA 2008, Montreal, Canada, February 25-28, 2008, Proceedings*, pages 158–168, 2008.
15. Sergei O. Kuznetsov. On the Intractability of Computing the Duquenne-Guigues Bas. *J. UCS*, 10(8):927–933, 2004.
16. Sergei O. Kuznetsov and Sergei A. Obiedkov. Counting Pseudo-intents and #P-completeness. In *Formal Concept Analysis, 4th International Conference, ICFCA 2006, Dresden, Germany, February 13-17, 2006, Proceedings*, pages 306–308, 2006.
17. Sergei O. Kuznetsov and Sergei A. Obiedkov. Some Decision and Counting Problems of the Duquenne-Guigues Basis of Implications. *Discrete Applied Mathematics*, 156(11):1994–2003, 2008.
18. Zbigniew Lonc and Miroslaw Truszczynski. On the Number of Minimal Transversals in 3-uniform Hypergraphs. *Discrete Mathematics*, 308(16):3668–3687, 2008.
19. Sebastian Rudolph. Some Notes on Pseudo-closed Sets. In *Formal Concept Analysis, 5th International Conference, ICFCA 2007, Clermont-Ferrand, France, February 12-16, 2007, Proceedings*, pages 151–165, 2007.

Edit Operations on Lattices for MDL-based Pattern Summarization

Keisuke Otaki[1,2] and Akihiro Yamamoto[1]

[1] Department of Intelligence Science and Technology,
Graduate School of Informatics, Kyoto University, Japan
ootaki@iip.ist.i.kyoto-u.ac.jp, akihiro@i.kyoto-u.ac.jp
[2] Research Fellow of the Japan Society for the Promotion of Science

Abstract. The closedness, a fundamental conception in FCA, is also important to address the pattern redundancy problem in pattern mining, where some enumerated patterns subsume others and they are redundant. Although many efficient mining algorithms have been proposed, finding characteristic and easy to interpret subsets of the whole enumerated patterns, called the *pattern summarization problem*, is still challenging. A well-used Information Theory-based criterion; the *Minimum Description Length* (MDL) principle helps us to tackle the problem, but it requires us to design the MDL evaluation from scratch according to types of patterns. In this paper we propose a new framework applicable to various patterns using lattices, which are beneficial to formulate the MDL evaluation. A key idea is revising an existing model by using *edit operations* defined on lattices among concepts on them, which enables us to consider additional information such as background knowledge and helps us to design the MDL evaluation. We experiment our method to see that our proposal helps us to obtain informative results from the whole sets, and confirm that our model is applicable to various patterns.

Keywords: formal concept analysis, lattice structures, edit operations, pattern summarization, minimum description length principle

1 Introduction

Knowledge Discovery using binary feature vectors is an important subject in data mining, where a binary value represents whether or not an object keeps some feature. A way to construct such vectors is using *pattern mining*: For a datum d and an enumerated pattern p, we compute a binary value $b_{d,p} = 1$ if and only if p occurs in d. Since the frequent itemset mining problem was first proposed by Agrawal and Srikant [2], various patterns and algorithms have been studied [1]. We can exploit obtained vectors as representations of data within several Machine Learning methods, and tasks such as clustering and classification can be tackled with the vectors. Therefore pattern mining is an essential tool.

Although many efficient mining algorithms have been studied, the *pattern redundancy problem* is still inevitable, where some patterns subsume others, enumerated patterns are not independent, and obtained vectors contain redundant

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors. Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

values. Considering the closedness, a main conception in FCA, is a fundamental way to resolve the problem and an example of the closedness used in pattern mining is closed itemsets [12], known as compact representations of itemsets.

The redundancy problem, however, still remains after considering the closedness only. To select a subset of the whole set of enumerated patterns, the *Minimum Description Length* (MDL) principle has attracted much attention [7, 8, 14–16, 18]. An important idea is to *represent databases with codes in a loss-less manner*, and to evaluate its difficulty by the length of codes used. For example, the Krimp algorithm [18] is a fundamental method for itemsets. Other methods for sequences and large graphs can be seen in [16] and [8], respectively. According to patterns, we need to design an evaluation method from scratch and it makes theoretical analyses difficult. It is also hard to integrate MDL-based methods with knowledge resources although various *side information* (e.g., corpora, ontology, etc.) are helpful to select enumerated patterns.

Developing methods applicable to various patterns in a unified framework and possible to consider side information is, therefore, an important but remaining problem. We in this paper present preliminary results of our new lattice-based framework based on an existing MDL-based method to enhance its utility for further developing of such methods to overcome problems above. Our key ideas are introducing edit operations on lattices (in Section 3.1) for the loss-less representation (by Proposition 1, Lemmata 1 and 2), and evaluating its difficulty by extending an existing formulation (in Section 3.2). To investigate the application possibility of our framework, we apply it to pattern concept lattices (in Section 4). We experiment them using common benchmark datasets (in Section 5). Our contributions are summarized below.

– *A new lattice-based framework*; it is extended from an existing model (code-tables in Section 2.2) by using terminologies from lattices.
– The *equivalency* for *closed itemsets* (by Proposition 1).
– *Pattern structured-based generalization*; *GenSet*, a generalization of itemsets using pattern structures (in Section 4) as an example of other lattices.
– *Empirical evaluation* using some benchmark datasets (in Section 5).

2 Preliminary

We outline Formal Concept Analysis (FCA) and Frequent Closed Itemset Mining (FCIM) for our new lattice-based framework, summarized in Table 1. We explain an idea of the MDL-based algorithm for itemsets, named the Krimp algorithm.

2.1 Basic Concepts

Formal Concept Analysis [5] Let $K = (G, M, I)$ be a *context*, where G and M are sets of objects and attributes, and $I \subseteq G \times M$ is a binary relation. A *Galois connection* between G and M, denoted shortly by $(\cdot)'$, is defined as $A' = \{m \in M \mid (g, m) \in I \text{ for all } g \in A\}$ for $A \subseteq G$ and $B' = \{g \in G \mid (g, m) \in$

Table 1: The correspondence between FCA and FCIM.

	FCA [5]	FCIM with a parameter θ [12]
objects	G; a set of objects	$\mathcal{D}$; a set of transactions
attributes	M; attributes	$\mathcal{I}$; items
functions	$\{(\cdot)', (\cdot)'\}$	$\{f, g\}$
task	find all concepts	find all itemsets $I \subseteq \mathcal{I}$ satisfying $freq(I) \geq \theta$

I for all $m \in B\}$ for $B \subseteq M$, respectively. A pair (A, B) is a *concept* if $A' = B$ and $B' = A$, where A and B are called the *extent* and the *intent*, and we denote them by $A = \mathrm{Ext}(c)$ and $B = \mathrm{Int}(c)$, respectively. For an object $g \in G$, the concept $(\{g\}'', \{g\}')$ is the *object concept* of g, denoted by γg.

Computing all concepts is an important task in FCA, and we denote the set of all concepts by $\mathfrak{B}(K)$. For two concepts (A, B) and (C, D), the partial order $\preceq$ is introduced by $A \subseteq C$, and $\langle \mathfrak{B}(K), \preceq \rangle$ becomes a complete lattice. We represent it by $\mathfrak{B}$ if it is understood from the context. For a concept $c \in \mathfrak{B}$, we define the *upset* of c, denoted $\uparrow c$, by $\{d \in \mathfrak{B} \mid c \preceq d\}$[3]. In addition we define the *direct upset* $\Uparrow c$ by $\{d \in \uparrow c \mid \text{there exists no } e \in \uparrow c \text{ s.t. } e \preceq d\}$.

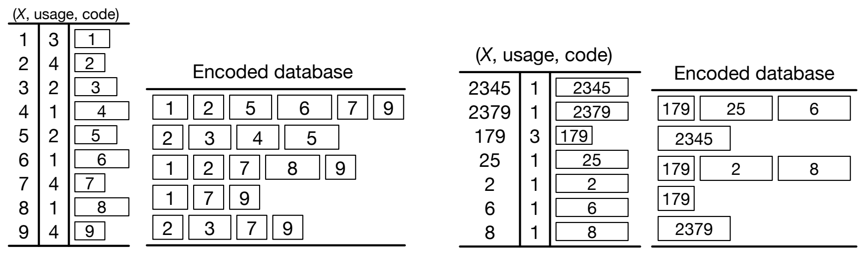
Frequent Closed Itemset Mining [12] Let $\mathcal{I}$ be the set of items. A set $I \subseteq \mathcal{I}$ of items is called an *itemset*, and it is also called a *transaction*. A *database* $\mathcal{D} = \{t_1, \ldots, t_N\}$ is a set of transactions. The *frequency* $freq(I)$ of an itemset I is defined as $freq(I) = |\{t \in \mathcal{D} \mid I \subseteq t\}|$. For a parameter θ, the frequent itemset mining (FIM) problem is the problem to find all itemsets satisfying $freq(I) \geq \theta$. We denote the set of all frequent itemsets by $\mathcal{F}$.

For *closed itemsets*, two functions f and g are defined as $f(T) = \{i \in \mathcal{I} \mid \forall t \in T, i \in t\}$ for $T \subseteq \mathcal{D}$ and $g(I) = \{t \in \mathcal{D} \mid \forall i \in I, i \in t\}$ for $I \subseteq \mathcal{I}$. An itemset I is *closed* if and only if $f(g(I)) = I$. The *frequent closed itemset mining* (FCIM) problem is to find all frequent and closed itemsets from $\mathcal{D}$ with respect to θ. It is easy to see that two functions f and g work in the same manner of $(\cdot)'$ in FCA[4]. We denote the set of all frequent *closed* itemsets by $\mathcal{CF}$.

The closed itemsets aim at reducing the number of all frequent itemsets enumerated in the *naïve* FIM problem, and many efficient algorithms have been studied [13, 17]. However the number of enumerated itemsets is still large in applications. They are often redundant because some patterns subsume others, which causes the *pattern redundancy problem*: *The enumerated patterns are too large and redundant to exploit them in applications although the closedness is considered.* In application, we often need only a small subset that characterizes all the enumerated itemsets, which are easy to interpret. We call such extraction problem *pattern summarization* (also known as *pattern set mining* [6, 15, 18]).

[3] This set $\uparrow c$ is also called the *principal filter* in partially ordered sets.

[4] We prepare *ids* of transactions and the set $\mathcal{D}' = \{(i_1, t_1), \ldots, (i_N, t_N)\}$. By constructing $I = \{(i_k, j) \mid 1 \leq k \leq N, j \in t_k\}$ from $\mathcal{D}'$, the FCIM problem of $\theta = 1$ is equivalent to computing all intents of concepts from $K = (\{i_1, \ldots, i_N\}, \mathcal{I}, I)$.



(a) The standard code-table ST (left) and the cover of $\mathcal{D}$ by ST (right).

(b) A code-table CT (left) and the cover of $\mathcal{D}$ by CT (right).

Fig. 1: The cover of $\mathcal{D} = \{125679, 2345, 12789, 179, 2379\}$ (from Example 1).

2.2 Two-part MDL Principle and The Krimp Algorithm

Out of various criteria for the selection, the *two-part MDL principle* [7] is often adopted for itemsets, where *code-tables* are well-studied models. Databases (or contexts) are encoded and represented by (binary) codes using code-tables.

Definition 1 (Code-tables [18]). A *code-table* CT consists of two columns for *itemsets* and *codes*. Formally $CT = \{(X_1, c_1), (X_2, c_2), \dots\}$ is a set of pairs, where $X_i \subseteq \mathcal{I}$, $c_i \in \mathbf{C}$, and $\mathbf{C}$ be the set of (binary) codes.

For the convenience of our discussion, in the following, we regard code-tables as *sets of itemsets*: For example $X \in CT$ means $(X, c_X) \in CT$ and $c_X$is denoted by $code_{CT}(X)$. We define $CT^- = \{(X, c_X) \in CT \mid |X| > 1\}$. The *standard code-table ST* consists of all *singleton itemsets* over $\mathcal{I}$. Let us first introduce examples, where sets are written as sequences (e.g. $\{1, 7, 9\}$ is represented as 179).

Example 1. Let $\mathcal{I} = \{1, \dots, 9\}$ and $\mathcal{D} = \{125679, 2345, 12789, 179, 2379\}$. In Fig. 1a and 1b, the left table is a code-table and the right one is a database represented by codes, where *rectangles* represent codes (e.g., the rectangle 7 means the code of the itemset 7). In Fig. 1a, the transaction 12789 is encoded separately. In Fig. 1b ($CT = ST \cup \{25, 179, 2345, 2379\}$), it is done by $179 \cup 2 \cup 8$.

The process of representing databases using code-tables is called the *cover*. Both frequent itemsets (e.g., 179) and non-frequent ones (e.g., 2345 or 2379) are used and the MDL principle takes a balance among them. The property of the cover of transactions is described by *cover functions.*

Definition 2 (Cover function[5]). Let $\mathcal{CT}$ be the set of all possible code-tables over $\mathcal{I}$, $\mathcal{D}$ be a database, $t \in \mathcal{D}$ be a transaction, and $CT \in \mathcal{CT}$ be a code-table. We call the set of itemsets $\{X \mid (X, c_X) \in CT\}$ the *coding set* of CT, denoted by CS. A *cover function* on CT, $cover : \mathcal{CT} \times 2^{\mathcal{I}} \to 2^{CS}$, should satisfy:

C1. if $X, Y \in cover(CT, t)$ then it holds either $X = Y$ or $X \cap Y = \emptyset$, and

[5] This definition is modified from Definition 2 of [18] by using a coding set CS.

C2. the union of all $X \in \mathit{cover}(CT, t)$ equals to t, i.e., $t = \bigcup_{X \in \mathit{cover}(CT,t)} X$.

Code-tables in Fig. 1 contain additional information *usage*. Using coding methods in the Kolmogorov Complexity [9], codes are decided according to *how often itemsets are needed*: For $X \in CT$, $\mathit{usage}(X)$ is defined as $|\{t \in \mathcal{D} \mid X \in \mathit{cover}(CT, t)\}|$, which determines the least length of prefix codes required.

Example 2 (from Example 1). In Fig. 1a, $\mathit{cover}(ST, 179) = \{1, 7, 9\}$, and in Fig. 1b $\mathit{cover}(CT, 12789) = \{179, 2, 8\}$. It holds that $\mathit{usage}(179) = 3$.

If a transaction $t \in \mathcal{D}$ is encoded by a cover function $\mathit{cover}(\cdot, \cdot)$ and some CT, we say that t is *covered*. Then the MDL principle can be evaluated.

Definition 3 (Lemma 1, Definitions 4 and 5 in [18]). The two-part MDL principle says that a code-table $CT \in \mathcal{CT}$ is the *best* for $\mathcal{D}$ if it minimizes $L(\mathcal{D}, CT) = L(CT) + L(\mathcal{D} \mid CT)$, where $L(CT)$ is the length of the code-table CT, and $L(\mathcal{D} \mid CT)$ is that of $\mathcal{D}$ when CT is given. Following Definitions 1 and 2, terms $L(CT)$ and $L(\mathcal{D} \mid CT)$ for $L(\mathcal{D}, CT)$ are evaluated as follows:

$$L(CT) = \sum_{X \in CT \text{ if } \mathit{usage}(X) \neq 0} L(\mathit{code}_{ST}(X)) + |\mathit{code}_{CT}(X)|, \tag{1}$$

$$L(\mathcal{D} \mid CT) = \sum_{t \in \mathcal{D}} \sum_{X \in \mathit{cover}(CT,t)} |\mathit{code}_{CT}(X)|, \tag{2}$$

where $L(\mathit{code}_{ST}(X)) = \sum_{i \in X} |\mathit{code}_{ST}(\{i\})|$.

The Krimp algorithm [18] is an iterative algorithm updating a code-table $CT \in \mathcal{CT}$ greedy by adding a new frequent itemset $X \in \mathcal{F}$ with evaluating $L(\mathcal{D}, CT)$. Please refer to Algorithm 3 in [18], which we follow in experiments.

2.3 Problems

Models based on code-tables have been widely used in MDL-based studies. The cover function adopts the set union to represent transactions, and the subset operation to divide them as seen in Example 1. The process assumes that items are in the same granularity, but it is not the case in applications. It also disables us to allow for side information (i.e., our background knowledge and resources) about databases as well. Then our targeting problems are listed below.

Problem 1. The cover function has a drawback of the difficulty of its generalization because it implicitly uses set operations.

Problem 2. Code-tables cannot reflect side information of items (e.g., *hierarchy*, or *degree of importance* of items) because of implicitly used set operations.

That is we need to revise set operations used in the existing method. Our key idea is to adopt terminologies of lattices to overcome these problems. In Section 3, we revise the cover function with lattices to build our new framework. We apply it to *pattern concept lattices* by *pattern structures* in Section 4 to confirm that our lattice-based framework is applicable to various patterns.

3 Models by Edit Operations for Closed Itemsets

We provide our framework based on lattices. For discussions, we define the object concept corresponding to $t \in \mathcal{D}$ by $\gamma t = (g(t), t)$ from the relation in Table 1. Here we only consider closed itemsets.

3.1 Cover Using Edit Operations on Lattices

In the cover of $t \in \mathcal{D}$, t is recursively replaced with $t \setminus B_1, t \setminus (B_1 \cup B_2), \ldots$ by choosing mutually disjoint closed itemsets $B_1, B_2, \ldots$ until t gets $\emptyset$ in some order[6]. We now have the following.

Observation 1 The cover of $t \in \mathcal{D}$ can be represented as a *sequence* of the *deletion* of items: $t \mapsto t \setminus B_1, t \setminus B_1 \mapsto t \setminus (B_1 \cup B_2), \ldots, t \setminus (B_1 \cup \cdots \cup B_m) \mapsto \emptyset$.

Thus a transaction $t \in \mathcal{D}$ can be covered by also using the deletion of items. We use such deletions to construct a new cover based on lattices. On lattices, selections of B_i corresponds to selecting concepts from $\mathfrak{B}$, and intents of them can be stored in code-tables. With respect to the search space of the selection, the following holds by properties of lattices.

Observation 2 For a transaction $t \in \mathcal{D}$, only a sub-lattice corresponding to the upset $\uparrow \gamma t$ of the corresponding object concept γt of t should be considered.

Choosing concepts, however, is *not complete* for the *loss-less property* (i.e., C2 of Definition 2) because in general $\mathfrak{B}$ does not contain all singleton itemsets in intents of concepts. In contrast, by the discussion of closed itemsets [12], all frequent itemsets could be recovered from closed itemsets and lattices should contain all information required.

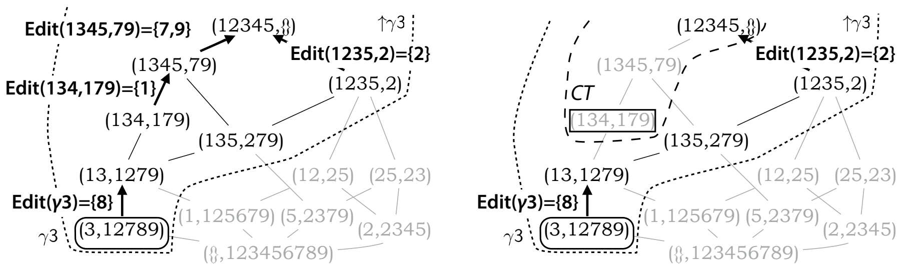
Our idea is extracting such hidden information on lattices as edit operations on lattice (e.g., the deletion of items in Observation 1), and represent $t \in \mathcal{D}$ by both the selections and edit operations. We first introduce *edit information.*

Definition 4 (Edit Information). For $c = (A, B) \in \mathfrak{B}$, the *edit information*, denoted by $\mathrm{Edit}(A, B)$, is defined as $\mathrm{Edit}(A, B) \equiv B \setminus \bigcup_{(C,D) \in \Uparrow (A,B)} D$. For a set $\mathcal{C}$ of concepts, it is extended by $\mathrm{Edit}(\mathcal{C}) = \bigcup_{(A,B) \in \mathcal{C}} \mathrm{Edit}(A, B)$.

For $c \in \mathfrak{B}$, $\mathrm{Edit}(c)$ indicates items which *cannot be represented by intents of concepts* except c itself even when all concepts in $\uparrow c \setminus \{c\}$ are chosen.

Example 3 (from Example 2). For $t_3 = 12789$, it holds that $\gamma 3 = (3, 12789)$ and $\mathrm{Edit}(\gamma 3) = \{8\}$. The item 8 *cannot be represented* by any concepts except $\gamma 3$. Therefore if $\gamma 3$ is not selected the item 8 *must be represented by another way.*

[6] The *standard cover order* is a basic one from the KRIMP algorithm: For an itemset $B \subseteq \mathcal{I}$, the *standard cover order* is the order of *the cardinality* $|B|$ *descending, the support freq*(B) *descending*, and *lexicographically ascending* [18]



(a) For $t_3 = 12789$ with ST (b) For $t_3 = 12789$ with CT

Fig. 2: Examples of the cover with edit operations for $\gamma 3 = (3, 12789)$ (i.e., t_3).

As another way, *edit operations* among intents of concepts are introduced to represent such information. Because $\mathfrak{B}$ contains the top $\top = (G, \emptyset)$[7] and $\mathrm{Int}(\gamma t) = t$, all information for covering t should be found in $\uparrow \gamma t$. We use edit information to capture such information and to explain edit operations on lattices (i.e., on intents of concepts). We encode edit information by tuples called *edit scripts*, and store them in *edit-tables* by analogy with code-tables.

Definition 5 (Edit scripts and Edit-tables). An *edit-table* ET consists of three columns: *kinds of operations*, arguments of operations (itemsets), and codes. That is, $ET = \{(t_1, X_1, c_1), \ldots, (t_m, X_m, c_m)\}$, where t_i indicates the type of operations, and both X_i and c_i is the same to code-tables. The tuple (t, X, c_X) is called an *edit script*. Let $\mathcal{E}$ be the set of all edit scripts.

Example 4 (from Example 3). Because $\mathrm{Edit}(\gamma 3) = \{8\}$, the item 8 should be represented by the deletion of the item 8. Let DEL represent the *deletion*[8]. A script $(\mathrm{DEL}, 8, c_8)$ with a code c_8 means the deletion of the item 8.

Definition 5 is general and abstract. For closed itemsets, however, we only need a specific kind of them. We define a function $\mathrm{Encode} : 2^{\mathcal{I}} \to 2^{\mathcal{E}}$ for $X \subseteq \mathcal{I}$ by $\mathrm{Encode}(X) = \{(\mathrm{DEL}, i, c_i) \mid i \in X, c_i \in \mathbf{C}\}$, where $\mathbf{C}$ is the set of codes.

Lemma 1. It holds that $\mathrm{Edit}(\uparrow \gamma t) = t$ for any $t \in \mathcal{D}$.

Proof If $\uparrow \gamma t = \{\gamma t, \top\}$, it holds that $\mathrm{Edit}(\uparrow \gamma t) = \mathrm{Edit}(\gamma t) = t$. Then $\mathrm{Encode}(\cdot)$ generates edit scripts of all items in t as the deletion of an item. By the induction on lattices (by $\Uparrow c$), we only need to confirm that for $c \in \mathfrak{B}$ all items which cannot be covered by intents of concepts in $\Uparrow c \setminus \{c\}$ can be represented by edit scripts. This is done by the definition of $\mathrm{Edit}(c)$ and properties of $\langle \mathfrak{B}(K), \preceq \rangle$. □

Lemma 1 guarantees that any $t \subseteq \mathcal{I}$ can be represented by using edit scripts when no concepts are chosen. Now the cover using lattices can be described using a function $cover_{\mathrm{L}}(\cdot, \cdot)$ by analogy with $cover(\cdot, \cdot)$. The following describes how to implement it instead of giving a new formal description of the function.

[7] If $B \neq \emptyset$ for the top $\top = (G, B)$ of $\langle \mathfrak{B}(K), \preceq \rangle$, we can insert a dummy $(G, \emptyset)$.

[8] The type is required to introduce several types of edit operations if needed.

Definition 6 (Revised Cover). The function $cover_{\mathrm{L}} : \mathcal{CT} \times 2^{\mathcal{I}} \to 2^{2^{\mathcal{I}}} \times 2^{\mathcal{E}}$ receives a code-table CT containing intents of selected concepts and a transaction t, and it returns a set of itemsets used (e.g., 179) and that of edit scripts.

For a code-table CT (See Fig. 2b), we first choose intents of concepts from CT in some order, and denote by $\mathcal{C}$ the set of selected concepts, which should be excluded from the search space $\uparrow\gamma t$. The remainder should be represented by edit scripts. Then $cover_{\mathrm{L}}(CT, t) = (\{\mathrm{Int}(c) \mid c \in \mathcal{C}\}, \mathrm{Encode}(\mathrm{Edit}(\uparrow\gamma t \setminus \uparrow\mathcal{C})))$.

Example 5 (from Example 3). It holds that $cover_{\mathrm{L}}(CT, t_3) = (\{179\}, \{(\mathrm{DEL}, 2, c_2), (\mathrm{DEL}, 8, c_8)\})$, representing the cover of t_3 in Fig. 2b. It holds that $cover_{\mathrm{L}}(ST, t_3) = \{\emptyset, \{(\mathrm{DEL}, 1, c_1), (\mathrm{DEL}, 2, c_2), (\mathrm{DEL}, 7, c_7), (\mathrm{DEL}, 8, c_8), (\mathrm{DEL}, 9, c_9)\}\}$, which simulates the cover of t_3 by singleton itemsets (in Fig. 2a).

If the set $\mathcal{C}$ in Definition 6 is the empty set $\emptyset$, the function $cover_{\mathrm{L}}(\cdot,\cdot)$ achieves the same result as the function $cover(\cdot,\cdot)$ with ST, supported by Lemma 1 (See Example 5). If some concepts are selected, the following lemma is important.

Lemma 2. Let $\mathcal{C}$ be the set in Definition 6 and $(\mathcal{X}, \mathcal{Y}) = cover_{\mathrm{L}}(CT, t)$. The set $\mathcal{Y}$ is complete to rerpesent $t \setminus \bigcup_{X \in \mathcal{X}} X$.

Proof From Definition 6, if a concept $c = (A, B)$ in $\uparrow\gamma t$ is selected and stored in CT (i.e., $c \in \mathcal{C}$), any item $i \in B$ is *not* represented by edit scripts in $\mathcal{Y}$ because there exists no concept $d \in \uparrow\gamma t \setminus \uparrow\mathcal{C}$ satisfying $i \in \mathrm{Edit}(d)$. It holds that $\{i \mid i \in \mathrm{Edit}(\uparrow\gamma t \setminus \uparrow\mathcal{C})\} = t \setminus \bigcup_{X \in \mathcal{X}} X$, and the function $\mathrm{Encode}(\cdot)$ ensures the lemma, i.e., the cover by $cover_{\mathrm{L}}(\cdot,\cdot)$ is valid when some concepts are chosen.

Together with the completeness in Lemma 1 and Lemma 2, our edit scripts are enough to represent any $t \in \mathcal{D}$ with selecting concepts in $\mathcal{C}$.

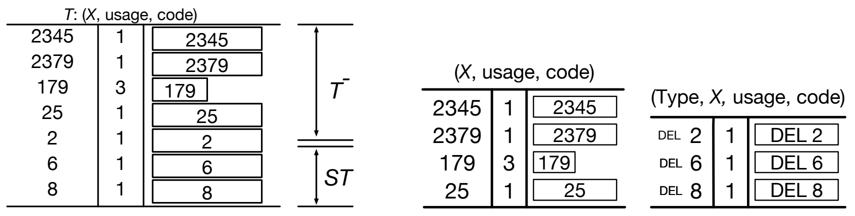
3.2 Model Evaluation Using Edit Operations

For our model (CT, ET), with reference to Section 2.2, the MDL principle $L(\mathcal{D}, CT, ET) = L(CT, ET) + L(\mathcal{D} \mid CT, ET)$ can be evaluated. For $L(CT, ET)$,

$$L(CT, ET) = \sum_{\substack{(X, c_X) \in CT \\ usage(X) \neq 0}} L(code_{ST}(X)) + |c_X| + \sum_{\substack{(t, X, c_X) \in ET \\ usage(X) \neq 0}} L(code_{\mathrm{OP}}(t)) + L(code_{ST}(X)) + |c_X|, \qquad (3)$$

where $code_{\mathrm{OP}}(t)$ represents codes used for distinguishing *types of edit scripts*. For itemsets, $L(code_{\mathrm{OP}}(t)) = 0$. The first $\sum$ term of Equation 3 is the same to the Krimp algorithm, and the second $\sum$ term is for encoding the edit-table ET.

The difficulty of representing databases, $L(\mathcal{D} \mid CT, ET)$, is evaluated by naturally generalizing the corresponding term of the Krimp algorithm below.



T: (X, usage, code)

2345	1	2345	T^-
2379	1	2379	
179	3	179	
25	1	25	
2	1	2	ST
6	1	6	
8	1	8	

(a) Code-table T $(= T^- \cup ST)$

(X, usage, code)

2345	1	2345
2379	1	2379
179	3	179
25	1	25

(Type, X, usage, code)

DEL	2	1	DEL 2
DEL	6	1	DEL 6
DEL	8	1	DEL 8

(b) Our model, two tables (CT, ET)

Fig. 3: A code-table T by the Krimp algorithm and our two tables (CT, ET). Our table CT is equivalent to T^- and ET simulates ST.

$$L(\mathcal{D} \mid CT, ET) = \sum_{t \in \mathcal{D}} \left(\sum_{(X, c_X) \in \mathcal{X}} |c_X| + \sum_{(t, X, c_X) \in \mathcal{Y}} |c_X| \right)$$
$$\text{for } (\mathcal{X}, \mathcal{Y}) = \mathit{cover}_{\mathrm{L}}(CT, t). \tag{4}$$

We illustrate an example in Fig. 3 to show how edit-tables intuitively work.

3.3 The Equivalency of Models for Closed Itemsets

For closed itemsets, we now have the following proposition.

Proposition 1 Let T be an obtained code-table by the Krimp algorithm for the database $\mathcal{D}$ with some order on $\mathcal{CF}$. With the same order, our model can achieve a pair (CT, ET) satisfying $L(\mathcal{D}, T) = L(\mathcal{D}, CT, ET)$ for closed itemsets.

Proof (*Sketch)* At the beginning, it holds that $T = ST$ (i.e., $T^- = \emptyset$) and $CT = \emptyset$, where all transactions should be covered by singleton itemsets. From the definition of Encode(·) and Lemma 1, the proposition immediately holds. If a closed itemset $B \in \mathcal{CF}$ is selected and stored as $B \in T$, there exists a concept $(A, B) \in \mathfrak{B}$ and its intent B is stored in our table CT. Because we use the same order in the cover of $t \in \mathcal{D}$, for $(\mathcal{X}, \mathcal{Y}) = \mathit{cover}_{\mathrm{L}}(CT, t)$, it holds that $\mathcal{X} = \mathit{cover}(T, t)$, meaning that the *usage* in T^- and CT must be identical. The *usage* of edit scripts must be the same as well by the fact $\mathcal{X} = \mathit{cover}(T, t)$, Lemmata 1, 2, and the definition of Encode(·). □

Summary Our idea is to revise the cover in the Krimp algorithm with lattices. Our framework divides code-tables into a part of intents of concepts (i.e., closed itemsets) and that of edit scripts simulating singleton itemsets. For *closed itemsets*, our model can achieve the same result as that obtained by the Krimp algorithm (Proposition 1), which are supported by properties of lattices.

The advantage of our lattice-based model is that lattices help us to generalize the MDL-based evaluation for various patterns. For example, *pattern concept* *lattices* [4] enable us to address the problem for several patterns with the MDL principle. The key concept for the generalization is to introduce edit operations on lattices according to patterns, encode them as edit scripts in edit-tables, where lattices become the search space in our framework. Lattices help us to consider the completeness of edit operations as well.

4 A Generalized Model via PSA

We apply our framework to *pattern concept lattices*. As an example of other lattices, we consider *GenSets*, a generalized version of itemset using *pattern structures*, which are introduced to analyze non-binary data [4]. We call methods using them for Knowledge Discovery *Pattern Structure Analysis* (PSA) below.

4.1 Pattern Structures and Generalized Sets

Pattern structures extend FCA using *meet semi-lattices*; a *meet semi-lattice* $(D, \sqcap)$ is a pair of a set D of *descriptions* representing fragments of objects as their features, and a meet operator $\sqcap$ between descriptions satisfying the associativity, the commutativity, and the idempotency. A partial order $\sqsubseteq$ of descriptions is induced by $\sqcap$ for $x, y \in D$ as $x \sqsubseteq y$ whenever $x \sqcap y = x$. A *pattern structure* $\mathbb{P}$ is a triple $\mathbb{P} = (G, (D, \sqcap), \delta)$, where G is a set of objects, $(D, \sqcap)$ is a meet semi-lattice, and a mapping $\delta : G \to D$ gives a descriptions for each object.

In PSA we use the following Galois connection $(\cdot)^\diamond$: $A^\diamond = \sqcap_{g \in A} \delta(g)$ for $A \subseteq G$ and $d^\diamond = \{g \in G \mid d \sqsubseteq \delta(g)\}$ for $d \in D$. A pair $(A, d) \in 2^G \times D$ is a *pattern concept* if and only if $A^\diamond = d$ and $d^\diamond = A$. A partial order among pattern concepts is defined as well, and *pattern concept lattices* can be constructed in a similar manner of FCA. Recent applications of PSA and lattices constructed by PSA can be seen in [3, 11]. We consider a slight modification of FCA by giving additional information to itemsets from the following scenario.

Example 6 (from Example 1). Let us consider $t_4 = 179$ of id i_4 and $t_5 = 2379$ of id i_5. We have $(i_4 i_5)' = 79$ computed by $i_4' \cap i_5' = 179 \cap 2379 = 79$. The remainders for each transaction are $1 = 179 \setminus 79$ and $23 = 2379 \setminus 79$, respectively. In this case, the existence of these items (i.e., 1 in t_4 and $2, 3$ in t_5) is *ignored* although they are useful for the representation of itemsets.

From this example, by using a meet semi-lattice, we introduce auxiliary numbers representing the *existence of items*, i.e., the number of items may be included at most in common, to enhance the expressiveness of itemsets.

Definition 7 (GenSets). We define $D = 2^{\mathcal{I}} \times \mathbb{N}$. For an itemset $B \subseteq \mathcal{I}$, we encode it by a pair $(B, 0) \in D$. The meet operator $\sqcap$ is defined as

$$(B_1, n_1) \sqcap (B_2, n_2) = (B_1 \cap B_2, \min(n_1 + |B_1 \setminus B_1 \cap B_2|, n_2 + |B_2 \setminus B_1 \cap B_2|)).$$

We call itemsets with additional numbers *GenSets* (*generalized sets* of items) [9].

Example 7. For two itemsets $t_4 = 179, t_5 = 2379$, we encode them as $d_1 = (179, 0)$ and $d_2 = (2379, 0)$, respectively. Then $d = d_1 \sqcap d_2$ is computed as $d = (B, n) = (79, 1)$, where $n = 1$ means that both itemsets t_4 and t_5 possibly have further one item. In other words, a GenSet $(79, 1)$ can be regarded as $\{7, 9, \star\}$ with a wildcard symbol $\star$ representing any item in $\mathcal{I}$.

We can interpret GenSets as a richer representation, and conjecture that they are helpful to capture characteristic features of itemsets. In other words, we now consider to take care of both the extent and the intent of concepts in evaluating the MDL principle. However, the Krimp algorithm cannot deal with GenSets directly because the symbol $\star$ is not in $\mathcal{I}$ and it has completely different meaning. Therefore, applying the ordinal cover is not suited and a new formulation is required from scratch. In contrast, our lattice-based model can be applicable. We can evaluate the two-part MDL principle for GenSets by only considering the cover function $cover_{\mathrm{L}}(\cdot, \cdot)$ using edit operations including $\star$.

4.2 The two-part MDL principle for GenSets

We can evaluate the two-part MDL principle only by considering the function $cover_{\mathrm{L}}(\cdot, \cdot)$ using edit operations based on pattern concept lattices.

Models for GenSets For a GenSet $(B, n) \in D$, we need to encode n to store GenSets in a code-table, which is possible by adding a column for storing the code for n. There exist several codes for numbers (e.g., arithmetic codes). We choose a simple one, which requires the length $-\log_2(1/P)$ bits where P is the number of possible candidates, and n is bounded by the minimum size of transactions in $\mathcal{D}$. Assuming this code, we set $P = |\mathcal{I}|$, the worst length of such representation.

The Cover with GenSets For $d = (B, n) \in D$, if $n = 0$, d is an itemset B. We focus on how to define the cover with GenSets when $n > 0$.

Example 8. Let $d = (B, n) = (79, 1)$ and $t_3 = 12789$. We first cover a part of t_3 by the itemset B and get $128 = t_3 \setminus B$. Additionally we *rewrite* a symbol $\star$ $(n = 1)$ for $\{1, 2, 8\}$. An item 8 is chosen and $code_{ST}(8)$ is used for this choice because the code of 8 in ST is longer than that of 1 and 2.

Please see Fig. 4 as well for the cover of t_3 using a GenSet $d = (79, 1) \in D$. As the symbol $\star$ subsumes any items in $\mathcal{I}$ we need to evaluate such subsumptions. To implement it, we assume that we would like to find general, common features which characterize many transactions by excluding non-frequent items as long

[9] For three sets $X, Y, Z \in 2^{\mathcal{I}}$ we want to compute $n = \min(|X| - |XYZ|, |Y| - |XYZ|, |Z| - |XYZ|)$, where $XYZ = X \cap Y \cap Z$, and we see that this value n can be computed by the counting and set operations.

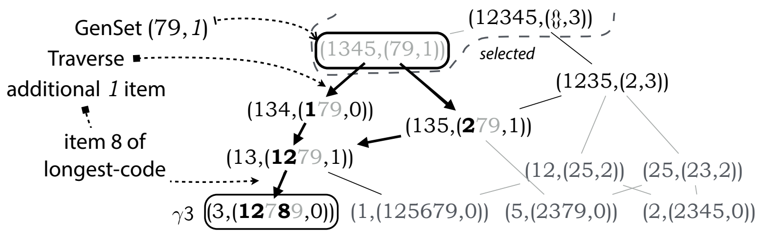


Fig. 4: For $t_3 = 12789$ with $CT = \{(79, 1)\}$, where $(79, 1)$ is a selected GenSet. An item can be covered by the GenSet $(79, 1)$ and we need to find it except 79.

as possible. We implement the subsumption of $\star$ as *replacing* the symbol $\star$ with $i \in \mathcal{I}$ by finding the item i having the longest $code_{ST}(i)$. Instead of giving a formal description, we give sketches of our idea below.

Sketch 1 We prepare a type SUB and construct an edit script as (SUB, i, c_i). As we have two types (DEL and SUB), $L(code_{\mathrm{OP}}(t))$ in Equation 3 is used. The function $\mathrm{Encode}(\cdot)$ should be revised with traversing of an item i having the longest code. For example in Fig.4, $cover_{\mathrm{L}}(\{(79, 1)\}, 12789)$ now should work like $(\{(79, 1)\}, \{(\mathrm{DEL}, 1, c_1), (\mathrm{DEL}, 2, c_2), (\mathrm{SUB}, 8, c_8)\})$ after traversing $i = 8$.

Sketch 2 For $d = (B, n)$, if $n > 0$ (i.e., the code of n shows non-zero value), we use a code $code_{ST}(\{i\})$ for representing the item i of the longest code in ST without edit scripts. This is done by extending the function $cover_{\mathrm{L}}(\cdot, \cdot)$, as a function like $cover^{new}_{\mathrm{L}}(\{(79, 1)\}, 12789) = (\{(79, 1)\}, \{(\mathrm{DEL}, 1, c_1), (\mathrm{DEL}, 2, c_2)\}, \{8\})$, for example, where $\{8\}$ can be immediately evaluated by ST computed from $\mathcal{D}$.

In dealing with pattern concept lattices, many ways are applicable to fulfill the loss-less property. Because both sketches are based on our edit-tables for itemsets in Section 3 and they adopt prefix codes, they achieve the loss-less property if they have codes for items subsumed by $\star$. As they have such codes in fact, the loss-less property holds for GenSets. In experiments we use Sketch 2.

Corollary 1 (The Loss-Less Property of GenSets) The loss-less property immediately holds for GenSets from discussions above.

We stress that lattices are compact representations of objects and their features, and they help us to prepare edit operations (i.e., edit scripts) and the cover function. Furthermore, we can introduce more labels in subsumptions of $\star$ for representing various information to take into account valuable side information in pattern mining. In addition, we expect that GenSets can be used not only for the symbol $\star$ but also for representing hierarchy by stacking GenSets as trees.

Table 2: Benchmark datasets represented in the form of contexts (G, M, I). Recall that $\mathcal{CF}$ is the set of all closed itemsets with $\theta = 1$.

	objects $\lvert G\rvert$	attributes $\lvert M\rvert$	$\lvert I\rvert/(\lvert G\rvert\lvert M\rvert)$	$\lvert\mathcal{CF}\rvert$
Iris	150	19	0.263	163
breast	699	16	0.624	641
pima	768	38	0.237	3,204
wine	178	68	0.206	14,555

5 Experiments

From the UCI Machine Learning repository [10], we use iris[10], breast[11], pima[12], and wine[13], which are also used in the Krimp algorithm [18] as benchmark datasets. All of those datasets are available from the distribution package[14] of the Krimp algorithm. The statistics of those datasets are summarized in Table 2.

All of our experiments were made on a machine of Mac OS X 10.10.2 with 2×2.26 GHz Quad-Core Xeon processors on 64GB main memory. All codes were written in Python 2.7.9. Our algorithm was the same as Algorithm 3 in the paper [18], which is an algorithm iteratively and greedy updating a code-table, except the part of the MDL evaluation. In our framework we replaced it with our implementation of Equations 3 and 4 based on lattices and edit operations. We did not discuss the computational complexity of our framework seriously. Rather than discussing solely run-times of experiments, we saw the followings.

- Comparison of two models: Results from two-models become the same by the equivalency of them (Proposition 1) for itemsets. We then investigate how much computational time our new model increases.
- We analyze the result with GenSets to check the application possibility of our lattice-based model for other patterns.

Let CT be the obtained code-table after applying the Krimp algorithm. For a database $\mathcal{D}$ we can compute both $L(\mathcal{D}, ST)$ and $L(\mathcal{D}, CT)$. We then measured the *ratio* $L(\mathcal{D}, CT)/L(\mathcal{D}, ST)$ (used in [18]), which can be regarded as a kind of the degree of compression achieved by CT, denoted **Ratio** in results. In addition we also measured the size $|CT^{-}|$ because it indicates how many concepts are chosen out of the whole set of enumerated patterns. Furthermore, in experiments with GenSets, we measured the *Jaccard index*: $J(X, Y) \equiv |X \cap Y|/|X \cup Y|$ for two sets X and Y, between a set of identifiers of closed itemsets $\mathcal{CF}$ and that of GenSets, to see the difference between results by itemsets and those by GenSets. Note that in our setting, the set of pattern concepts by GenSets can be regarded as a set of closed itemsets if we ignore additional numbers n in $(B, n) \in D$.

[10] https://archive.ics.uci.edu/ml/datasets/Iris
[11] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[12] https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
[13] https://archive.ics.uci.edu/ml/datasets/Wine
[14] http://www.patternsthatmatter.org/software.php, confirmed Feb. 2015.

Table 3: Results for itemsets by code-tables (the Krimp algorithm) and two-tables (concepts and edit-tables in our framework).

	$\lvert\mathcal{CF}\rvert$	$\lvert CT^{-}\rvert$	**Ratio**	**Krimp** Pre-proc. [s]	**Krimp** Run-time [s]	**I-Edit** Pre-proc. [s]	**I-Edit** Run-time [s]
iris	163	**13**	0.456	0.025	0.234	0.021	0.520
breast	641	**29**	0.181	0.580	6.263	0.573	26.794
pima	3,204	**66**	0.356	18.202	47.525	19.107	*373.63*
wine	14,555	**72**	0.757	442.686	393.539	444.842	*1354.570*

In results, we labeled by **Krimp** to show results by the Krimp algorithm, by **I-Edit** to show results by our framework using lattices and edit operations, and by **Gen-Edit** for results by GenSets using pattern concept lattices.

5.1 Results and Discussions for Itemsets

We show in Table 3 results of experiments for itemsets based on the existing method **Krimp** and our framework **I-Edit**, where we focus on run-times only. Note that we, of course, confirmed that results are equivalent.

With respect to the time for pre-processing, there existed no large differences. Concerning the time for the iterative updating process of models, roughly speaking our new model is at most 10 times slower than code-tables. Because both models are not implemented by sophisticated manners, we conjecture that our implementation cannot exploit pre-processed information on lattices well. That is, computing edit-tables using lattices is time-consuming and not sophisticated. Because the part of edit operations apparently depends on implementations of lattices and data structures for concepts, we expect that some advanced data structures help us to overcome this bottleneck.

5.2 Results and Discussions for GenSets

We conducted the same experiments using the same datasets with GenSets (in Section 4.1) and our MDL evaluation (in Section 4.2). Results are summarized in Table 4. We can see that the numbers of selected concepts via GenSets (i.e., 12, 24, 62, and 40) are *lower* than those via itemsets (i.e., 13, 29, 66, and 72), which mean that obtained models via GenSets of databases are *more compact* than those via itemsets. The same trend can be confirmed in the **Ratio** column.

The run-times of experiments have become faster than those in Table 3. Because the numbers of concepts (itemsets) and pattern concepts (GenSets) are the same now, these results should arise from the differences of the cover and the MDL evaluation. In addition the contents of the selected patterns should be affected. We can see the difference of the contents of the selection by checking the Jaccard index. As we can see that, our GenSets formulation surely generates *completely* different results compared with the results for itemsets.

Table 4: Results by our lattice-based framework for GenSets.

		Gen-Edit				
	$\lvert\mathcal{CF}\rvert$	$\lvert CT^{-}\rvert$	**Ratio**	Pre-proc. [s]	Run-time [s]	Jaccard Index
IRIS	163	**12**	0.428	0.009	0.550	0.785
BREAST	641	**24**	0.168	0.351	20.666	0.293
PIMA	3,204	**62**	0.342	18.92	285.676	0.123
WINE	14,555	**40**	0.686	415.376	454.631	0.131

Furthermore, the existence of stars $\star$ in GenSets affect the results. For GenSets (in Section 4.2), we aggressively cover items having longer codes in ST if the symbol $\star$ appears. As less frequent itemsets have longer codes in the table ST and such items are covered by $\star$ in advance, concepts stored in our model have relatively high frequency in the database, and the results get more compact.

Concluding Remarks on Experiments, and Future Directions We conducted the MDL-based pattern summarization for selected benchmark datasets by using code-tables and our model (two tables). We also applied our model for GenSets to which we can apply the same framework. At least we confirmed that our model obtained more compact results using GenSets (checked by **Ratio**), and the contents are also different (done by Jaccard index).

With respect to the *quality of the choice* of concepts via the two-part MDL principle it is always open to discuss. To discuss the quality, the previous work [18] developed the classifier using code-tables, tested it for labeled data, and discussed the quality of their choice. Therefore following their strategy and developing a classifier using edit operations and lattices are our future work to consider the pros and cons of our model and the quality of the choice.

Further discussion on the relation of run-times for edit-tables and properties of lattices (e.g., the density of contexts), is also important. Because the order of checking the set $\mathcal{CF}$ is optional, developing and analyzing further heuristic based on lattices is also important. We expect that several terminologies (not used in this paper) and methods for lattices would be helpful for this direction.

6 Conclusions and Future Work

In this paper we propose a new framework for pattern summarization to conquer the redundancy problem for various patterns by combining lattices and edit operations, which are evaluated by the two-part MDL principle. Our experiments show that our model successfully extends an existing model towards more general patterns. As an example we confirm that our model is applicable to GenSets, defined by pattern structures as a richer representation of itemsets. For closed itemsets, our model achieves the same results as the existing model (Proposition 1). Results of **Ratio** in experiments, GenSets are useful to obtain more compact representations of database compared with an existing model.

In our future work, we plan to develop a classifier based on the MDL principle by following the existing work for discussing the quality of the selection. We also investigate encoding methods and heuristic for the cover towards various patterns together with side information. Theoretical analysis of the two-part MDL evaluation via lattices is also our important future research direction.

Acknowledgements. The authors are grateful to the anonymous reviewers for valuable comments. This study was partially supported by Grant-in-Aid for JSPS Fellows (26-4555) and JSPS KAKENHI Grant Number 26280085.

References

1. Aggarwal, C.C., Han, J. (eds.): Frequent pattern mining. Springer International Publishing (2014)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. of the 20th VLDB. pp. 487–499 (1994)
3. Buzmakov, A., Egho, E., Jay, N., Kuznetsov, S.O., Napoli, A., Raïssi, C.: The representation of sequential patterns and their projections within formal concept analysis. In: Workshop Notes for LML (ECML/PKDD2013) (2013)
4. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Proc. of the 9th ICCS. pp. 129–142 (2001)
5. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
6. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling databases. In: Proc. of the 7th DS. pp. 278–289 (2004)
7. Grünwald, P.D.: The minimum description length principle. The MIT Press (2007)
8. Koutra, D., Kang, U., Vreeken, J., Faloutsos, C.: VOG: Summarizing and understanding large graphs. In: Proc. of the 22nd SDM. pp. 91–99 (2014)
9. Li, M., Vitnyi, P.M.: An introduction to kolmogorov complexity and its applications. Springer Publishing Company, Incorporated, 3 edn. (2008)
10. Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
11. Otaki, K., Ikeda, M., Yamamoto, A.: Pattern structures for understanding episode patterns. In: Proc. of the 11th CLA. pp. 47–58 (2014)
12. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Prof. of the 7th ICDT. pp. 398–416 (1999)
13. Pei, J., Han, J., Mao, R.: CLOSET: An efficient algorithm for mining frequent closed itemsets. In: Proc. of the DMKD2000. pp. 21–30 (2000)
14. Smets, K., Vreeken, J.: Slim: directly mining descriptive patterns. In: Proceedings of the 20th SDM. pp. 236–247 (2012)
15. Tatti, N., Vreeken, J.: Discovering descriptive tile trees. In: Proc. of the ECML/PKDD 2012. pp. 24–28 (2012)
16. Tatti, N., Vreeken, J.: The long and the short of it: Summarising event sequences with serial episodes. In: Proc. of the 18th KDD. pp. 462–470 (2012)
17. Uno, T., Asai, T., Uchida, Y., Arimura, H.: LCM: An efficient algorithm for enumerating frequent closed item sets. In: Proc. of the FIMI'03 (2003)
18. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: Mining itemsets that compress. Data Mining and Knowledge Discovery 23(1), 169–214 (2011)

An Approach Towards Classifying and Navigating RDF data based on Pattern Structures

Mehwish Alam and Amedeo Napoli

LORIA (CNRS – Université de Lorraine)
BP 239, 54506 Vandoeuvre-les-Nancy, France
{mehwish.alam,amedeo.napoli@loria.fr}

Abstract. With an increased interest in machine processable data, more and more data is now published in RDF (Resource Description Framework) format. This RDF data is present in independent and distributed resources which needs to be centralized, navigated and searched for domain specific applications. This paper proposes a new approach based on Formal Concept Analysis (FCA) to create a navigation space over semantic web data. This approach uses an extension of FCA and takes RDF triples and RDF Schema present on several independent sources and provide centralized access over the data resulting from several resources. Afterwards, SPARQL queries can be posed over this navigation space to access these distributed resources from one platform for information retrieval purposes.

Keywords: Formal Concept Analysis, Navigation Space, Linked Open Data, RDF, RDF Schema, Semantic Pattern Structures.

1 Introduction

With the efforts of Semantic Web community many technologies have been offered for publishing machine-readable data on web. It annotates textual data with meta-data and makes it available in the form of ontologies and RDF graphs. One of the emerging source of such data are published in the form of Linked Open Data (LOD) cloud [2]. As a contrast to textual resources, LOD does not need extensive preprocessing as it is already annotated in the form of entities and relationships.

This structured format leads to other kind of challenges. One of the basic characteristics of Semantic Web is that it follows a *decentralized* publication model, meaning that the RDF graphs are published in several distributed resources, instead of creating one knowledge-base of statements any one can contribute new statements and make it publicly available. These resources have nothing in common except some shared terms. These decentralized graphs should be integrated through machine/software agents to provide domain specific applications. Moreover, external schemas in the form of ontologies or taxonomies

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors.
Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

can be linked to these data to make sense based on real world conception. Some of the resources may only consist of ontological information and may not contain the instantial information such as SWRC ontology [11] and some resources may only contain instantial information such as DBLP. The problem of how to provide one platform for accessing several and related semantic web resources to build domain specific applications still persists. This paper focuses on how to link several related heterogeneous RDF resources present on distributed locations containing RDF data and RDF Schema into one space which can further be navigated and searched by the end user. We call it a "navigation space" because it provides centralization over RDF triples and also over RDF Schema belonging to several resources. This space provides navigation and querying over RDF triples as well as RDF Schema.

The current paper introduces a new framework based on Formal Concept Analysis [8] which focuses on how a navigation space is created over RDF data by taking into account an existing RDF Schema to provide search capabilities over distributed and heterogeneous Semantic Web resources. In the current study we only consider a part of RDF Schema which keeps subclass-superclass relation. Other constructs such as sub-property and relation between classes are not considered here. FCA alone may not be able to handle the complex data such as RDF data and RDF Schema as it considers only binary data. To deal with this shortcoming, we employ an extension of FCA, namely Pattern Structures [7]. We propose a new framework called Semantic Pattern Structures, that takes RDF triples and the RDF Schema as an input and produces a "navigation space". This *navigation space* provides centralization over distributed RDF resources and keeps a partially ordered organization of the classes of RDF triples with respect to RDF Schema from data sources which may or may not be part of LOD. The lattice structure allows for querying over RDF graph w.r.t. subject, predicate and object and makes use of the partial order relation between the concepts. This enables simultaneous search over schema and the facts contained in an RDF graph. This navigation space allows querying using well-documented and well-know query language, SPARQL. Several data sources are simultaneously queried through one platform. To date this is the first attempt to deal with RDF graph and RDF Schema using pattern structures.

The paper is structured as follows: section 2 gives the motivating example, section 3 introduces the basic notion of Pattern Structures with an example. Section 4 details the framework of semantic pattern structures for creating schema rich navigation space over RDF data. Section 5 defines the querying mechanism over the obtained navigation space. Section 6 describes the experimental results of the framework. Section 7 discusses the related work while Section 8 concludes the paper.

2 Motivating Example

For instance, if a patient has been prescribed some Cardiovascular Agent (CVA) and after a while he is looking for a replacement for his drug because it causes some allergic condition as a side effect. For finding solution, the doctor should be able to access this information for obtaining the suggestions for replacing patient's drug which is used for the same purpose but does not cause the side effect reported by the patient. All the required information is present in different data sources:

- Drugbank[1] contains information about drugs with their categories and the proteins it targets.
- SIDER[2] contains drug with their specific side effects.
- The category of drugs CVA exists in MeSH [3].
- Allergic conditions which is a class of side effects exists in MedDRA[4].

The information required by the domain expert cannot be obtained by posing a simple query on Google. SPARQL queries over each resource should be written to obtain this answer, which further needs manual work to obtain the required suggestions as the answers are in the form of list. In order to access such kind of information, all the data sets must be accessed simultaneously and there should be a single information space for accessing all the desired information through navigation and simple SPARQL queries.

In the current study, we propose a generic framework that provides one space for accessing several data sources related to some domain. The framework proposed in this study is generic because it can be applied to any domain having related information scattered over several resources. Based on the requirements of the user and the applications, the navigation space can be defined on a limited number of factual RDF triples. For example, in case of a cardiologist, he will only be interested in Cardiovascular Agents or some related categories of drugs. This way the application will include the drugs which are Cardiovascular Agents and its related categories. Only a subset of datasets should be considered according to the requirements of the applications. These specification are described by the domain expert as a starting point for narrowing down the huge cloud of Linked Data present on-line.

3 Preliminaries

3.1 Linked Open Data

Recently, Linked Open Data (LOD) [2] has become a standard for publishing data on-line in the form of Resource Description Framework (RDF)[5] which can further be linked to other data sources published in the same format. The idea underlying RDF is based on storing statements about a particular resource, where each statement is represented as $\langle subject, predicate, object \rangle$. A

[1] `http://www.drugbank.ca/`
[2] `http://sideeffects.embl.de/`
[3] `http://www.ncbi.nlm.nih.gov/mesh/`
[4] `http://meddra.org/`
[5] http://www.w3.org/RDF/

set of linked statements is referred to as RDF graph which constitutes an RDF triple store. For instance, Table 2 keeps RDF triples for drugs with their side effects and drug categories. The prefixes and full forms of all the abbreviations used in this paper are shown in Table 1. Consider $t1$ i.e., $\langle s_1, p_1, o_1\rangle$, here s_1 is subject, p_1 is predicate and o_1 is the object. A URI $\mathbf{U}$ in an RDF graph is a general form of URL. The actual representation of the drug s_1 in the form of URI is $'$`http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/drugs/DB00669`$'$, which is a URI of the DrugBank provided by University of Mannheim containing all the information about the drug in the form of triples. *DB00669* is the id provided by DrugBank to drug s_1 and the name of the drug is obtained by using the `rdfs:label` predicate defined in RDFS vocabulary[6]. The subject denotes the *resource* and the predicate denotes properties of the resource and defines relationship between the subject and the object. In the rest of the paper we use dereferenced resources i.e., s_1 instead of complete URI.

Practically, we consider RDF data w.r.t. three levels according to the value of predicate.

– An RDF triple is at the *schema level* when the predicate has a reserved value from RDFS vocabulary such as `rdfs:subClassOf, rdfs:Class` (represented as `sc` and `Class` respectively in the rest of the paper). The keyword `rdfs:Class` is used to declare that a set of resources is constituting a class and `rdfs:subClassOf` states that all the instances of one class are the instances of another class. For example, in Table 2, the *schema level* is given by triples $t6, t8$. An example of the tree structure contained in an RDF schema (in the rest of the paper when we use RDF Schema/schema we mean the tree structure created by `rdfs:subClassOf`) related to the drug side effect and their categories is shown in Figures 1 and 2 respectively.
– An RDF triple is at the *type level* if it keeps the mappings from an instance to its class and states that a resource is an instance of a class. The RDF triple is at this level when the predicate has a reserved value from RDF vocabulary such as `rdf:type` (represented as `type` in the rest of the paper). For example, in Table 2, the *type level* is given by triples $t4, t5, t7$.
– An RDF triple is at *factual level* when the predicate is related to the domain knowledge and is not a keyword from RDF vocabulary. For example, in Table 2, the *factual level* is given by $t1, t2, t3$.

Table 2 keeps RDF triples from several resources. The information about the side effect of the drugs shown in triples $t1, t2$ is present on SIDER[7] database which keeps the side effects of the drugs contained on the packaging of the drug. The information about the categories of the drugs shown in triples $t3$ is present on DrugBank[8], which keeps detailed information about each of the drugs and the proteins it targets. The schema level information about the drug side effects shown in $t5, t6$ are present in The Medical Dictionary for Regulatory Activities

[6] http://www.w3.org/TR/rdf-schema/
[7] http://sideeffects.embl.de/
[8] http://www.drugbank.ca/

Abbreviation	Term	Abbreviation	Term
p_1	`http://wifo5-04.informatik.uni-mannheim.de/sider/resource/sider/sideEffect`	p_2	`http://wifo5-04.informatik.uni-mannheim.de/drugbank/resource/drugbank/category`
s_1	Sumatriptan	C_1	Acute Coronary Syndrome
s_2	Tadalafil	C_4	Setevens-Johnsons Disorders
s_3	Theophylline	C_2	Tachycardia
s_4	Ticlopidine	C_3	Urticaria
s_5	Vardenafil	C_5	Erythmia Multiforme
D_1	Fibrinolytic Agents	C_6	Coronary Artery Disorders
D_2	Vasoconstrictor Agents	C_7	Cardiac Arrhythmias
D_3	Vasodilator Agents	C_8	Urticarias
D_4	Fibrin Modulating Agents	C_9	Allergic conditions NEC
D_5	Cardiovascular Agents	C_{10}	Allergic conditions
D_6	Molecular Mechanisms of Pharmacological Action	C_{11}	Cardiac Disorders
D_7	Therapeutic Uses	C_{12}	Immune System Disorders

Table 1: This table keeps the abbreviations of the terms used in the rest of the paper.

tid	Subject	Predicate	Object	Provenance
t1	s_1	p_1	o_1	SIDER
t2	s_1	p_1	C_2	SIDER
t3	s_1	p_2	D_2	DrugBank
t4	s_1	`type`	Drug	DrguBank
t5	o_1	`type`	C_1	MedDRA
t6	C_1	`sc`	C_6	MedDRA
t7	D_2	`type`	D_5	MeSH
t8	D_5	`sc`	D_7	MeSH
⋮	⋮	⋮	⋮	⋮

Table 2: RDF triples for the Drug Domain from several resources.

(MedDRA) Terminology[9] which is the international medical terminology. Finally the schema level information related to the drug category shown in triples $t7, t8$ is present in MeSH (Medical Subject Headings)[10] is a controlled vocabulary thesaurus which along with other tree structures keeps classification of drug categories.

3.2 SPARQL

A standard query language for accessing RDF graphs is SPARQL[11] which mainly focuses on graph matching. A SPARQL query is composed of two parts the head and the body. The body of the query contains the Basic Graph Patterns (it is contained in the WHERE clause of the query). It is composed of complex graph patterns that may include RDF triples with variables, conjunctions, disjunctions and constraints over the values of the variables. These graph patterns are matched against the RDF graph and the matched graph is retrieved and manipulated according to the conditions given in the query. The head of the query is an expression which indicates how the answers of the query should be constructed. For instance, consider a simple SPARQL query `SELECT ?x ?y WHERE {` `?x` p_1 `?y}`, then the basic graph pattern `?x` p_1 `?y` will be matched against the triples containing p_1 as predicate i.e., $t1$ and $t2$ (see Table 2). Thus the answer of the query will be (s_1, o_1), (s_1, C_2).

[9] http://www.meddra.org/
[10] http://www.ncbi.nlm.nih.gov/mesh
[11] http://www.w3.org/TR/rdf-sparql-query/

3.3 Pattern Structures

Formal Concept Analysis [8] can process only binary context, more complex data such as graphs, RDF triples can not be directly processed by FCA. Some of the complex data require scaling for these to be considered as binary data. The concept lattice obtained by a binary context mixes between several types of attributes. Pattern structures [7], an extension of FCA, allows direct processing of such kind of complex data such as RDF triples (as we show in this paper). The pattern structures were introduced in [7].

A pattern structure is a triple $(G, (D, \sqcap), \delta)$, where G is the set of entities[12], $(D, \sqcap)$ is a meet-semilattice of descriptions D and $\delta : G \to D$ maps an entity to a description. More intuitively, a pattern structure is the set of objects with their descriptions with a similarity operation $\sqcap$ on them which represents the similarity of objects. This similarity measure is idempotent, commutative and associative. If $(G, (D, \sqcap), \delta)$ is the pattern structure then the derivation operators can be defined as:

$$A^{\square} := \bigsqcap_{g \in A} \delta(g) \qquad for\ A \subseteq G$$

$$d^{\square} := \{g \in G | d \sqsubseteq \delta(g)\} \qquad for\ d \in D$$

Now the pattern concept can be defined as follows:

Definition 1 (Pattern Concept). *A pattern concept of a pattern structure "$G, (D, \sqcap), \delta$" is a pair (A, d) where $A \subseteq G$ and $d \in D$ such that $A^{\square} = d$ and $A = d^{\square}$, where A is called the concept extent and d is called the concept intent.*

Finally, the obtained concept lattice is called as *pattern concept lattice*, which keeps partially ordered relations between the pattern concepts.

4 Towards Semantic Pattern Structures

The *decentralization* issue is raised when many entities are contributing statements and making it publicly available. Either they are publishing same data using different vocabularies or related data which is distributed over several resources. To provide centralized access over distributed resources this section discusses a framework called Semantic Pattern Structures. This new framework is called Semantic Pattern Structures (SPS) because it is based on Pattern Structures and it deals with data in the form of triples and it classifies these triples with respect to RDF Schema present in the form of taxonomy. This way the final navigation space also keeps the semantic information i.e., background knowledge. The idea underlying the SPS is to create a concept lattice directly from an RDF graph and use the associated RDF Schema present on Linked Open Data to

[12] We rename an object in Pattern Structures as an *entity* to avoid confusion with the object in an RDF triple.

be accessed within one concept lattice. This concept lattice provides search and navigation capabilities over RDF data as well as over the RDF Schema to answer certain queries of the domain expert. The obtained lattice does not only provides access to several data sources but is also a materialization of the associated RDF Schema.

For creating one navigation space over RDF as well as RDF Schema, the first task is to define RDF triples in terms of patterns structures i.e., specifying the entities, their descriptions and the mapping from entities to description. The second task is to select a suitable RDF Schema associated to these RDF triples from distributed data sources based on domain knowledge. After these two preprocessing steps, we define a similarity measure $\sqcap$ over two descriptions which we generalize to the set of descriptions. After defining the similarity measure, we explain how a semantic pattern concept lattice is built using this similarity measure. Finally, we interpret, provide navigation and querying capabilities over the obtained pattern concept lattice.

4.1 From RDF Triples to Semantic Pattern Structures

Firstly, we represent factual RDF triples about certain domain as entities and their descriptions. According to section 3.3, pattern structures are given as $(G, (D, \sqcap), \delta)$. The descriptions D are termed as semantic descriptions and are denoted as D_s. An RDF triple store containing factual level information consists of statements of the form $\langle s_i, p_j, o_k \rangle$, where $i = \{1, \ldots, n\}, j = \{1, \ldots, q\}, k = \{1, \ldots, m\}$. Then, the set of subjects s_i are mapped to the entities in G. As the set of entities G represent the subjects in triples, we represent it as S.

Regarding the type of each object set which is the range of same predicate a suitable RDF Schema is obtained by locating the terms in the closely related RDF Schema. This selection is done based on domain knowledge. RDF Schema contains many constructs such as property, sub-property etc. along with `sc` and information but in this work we use the RDF Schema predicates such as `type` and `sc` introduced in section 3.1. First, the mapping from object to its type is obtained such as $\langle o_k, \texttt{type}, C_1 \rangle$ i.e., o_k is an instance of class C_1. Each object is then replaced by its corresponding class i.e., o_k is replace by C_1. Now, the considered taxonomy information used for defining the similarity measured is present in the RDF Schema by following the predicates such as $rdfs : subClassOf, skos : broader$ e.g., $\langle C_1, \texttt{sc}, C_6 \rangle$, i.e., C_1 is subclass of C_6. Note that we do not extract the trees, we follows this predicate while defining the similarity in the next section.

For instance, to obtain generic drug categories such as cardiovascular agents or generic side effects such as allergic conditions we further add the class information related to each of the objects in the triples. For example, *(*o_k*,*`type`*,*C_2*)* meaning that o_k is an instance of class C_2. Afterwards, we select all the super classes of the current class in the RDF Schema. For example, for C_2, we have C_7, C_{11} and C_{13}. An RDF Schema for drug side effect and drug category is shown in Figure 1 and Figure 2 respectively.

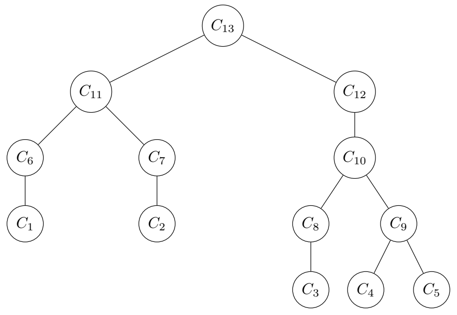


Fig. 1: RDF Schema for Side Effects (MedDRA) ($\mathcal{T}_1$).

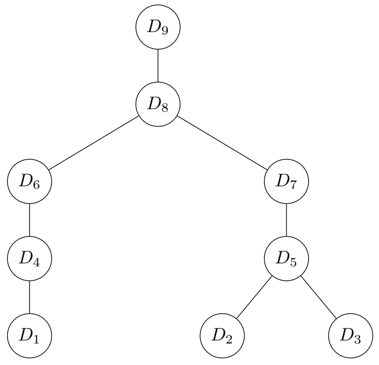


Fig. 2: RDF Schema for Drug Category (MeSH) ($\mathcal{T}_2$).

The pair $(p_j : C_k)$ gives a description $d_{ij} \in D_s$. The mapping of entities to description $\delta : S \rightarrow D_s$ is given as follows: let $s_i \in S$ then $\delta(s_i) = \{d_{i1}, \dots, d_{iq}\}$ where $i \in \{1, \dots, n\}$ and $d_{ij} = \{p_j : \{C_1, C_4, \dots, C_m\}\}$ where $j \in \{1, \dots, q\}$. Finally, the semantic pattern structures are given as $(S, (D_s, \sqcap), \delta)$. Consider the running scenario described before. As the drugs in the DrugBank and SIDER are same and are the subjects then the triples $t1, t2, t3$ in Table 2 are represented as $(S, (D_s, \sqcap), \delta)$ where the entity S has the description $\delta(s_1) = \{$ *(*$p_1 : \{C_1, C_2, C_3\}$*)*, *(*$p_2 : \{D_2\}$*)* $\}$. The descriptions are based on same subjects having different descriptions belonging to different resources i.e., side effect and category. Finally, with respect to the triples in Table 2, the subjects in the RDF triples are mapped to the entities, $S = \{s_1, s_2, \dots\}$, the descriptions are given as $D_s = \{$*(*$p_1 : \{C_1, C_2, C_3, \dots\}$*)* , *(*$p_2 : \{D_1, D_2, \dots\}$*)*$\}$.

Entities S	Descriptions D_s
s_1	$\{(p_1 : \{C_1, C_2, C_3\}), (p_2 : \{D_2\})\}$
s_2	$\{(p_1 : \{C_1, C_4, C_5\}), (p_2 : \{D_3\})\}$
s_3	$\{(p_1 : \{C_2\}), (p_2 : \{D_3\})\}$
s_4	$\{(p_1 : \{C_3, C_4, C_5\}), (p_2 : \{D_8\})\}$
s_5	$\{(p_1 : \{C_1, C_3\}), (p_2 : \{D_2\})\}$

Table 3: RDF Triples as entities S and semantic descriptions D_s.

4.2 Similarity Operation ($\sqcap$) over Descriptions

For defining the similarity operation over the semantic description D_s, we use the notion of least common subsumer (lcs). Here we re-write the definition of a least common subsumer of two classes in a RDF Schema $\mathcal{T}$ given in [9].

Definition 2 (Least Common Subsumer). *Given a partially ordered set* $(S, \leqslant)$*, a least common subsumer* E *of two classes* C *and* D *(lcs(C,D) for short) in a partially ordered set is a class such that* $C \sqsubseteq E$ *and* $D \sqsubseteq E$ *and* E *is least i.e., if there is a class* E' *such that* $C \sqsubseteq E'$ *and* $D \sqsubseteq E'$ *then* $E \sqsubseteq E'$.

Example 1. Let us consider C_3 and C_5. Then according to the RDF Schema in Figure 1, the common subsumers of C_3 and C_5 are C_{10}, C_{12}, C_{13} and $lcs(C_3, C_5) = C_{10}$.

Similarity Operation over Set of Descriptions: For describing the similarity over set of descriptions, LCS is computed pairwise for classes in two different sets of description and then only the most specific classes are picked.

Let s_1 and s_2 be two entities such that $s_1, s_2 \in S$ and the mapping δ is given as follows $\delta(s_1) = d_1 = \{p_1 : \{C_1, C_2, C_3\}\}$ and $\delta(s_2) = d_2 = \{p_1 : \{C_1, C_4, C_5\}\}$. The similarity is always computed w.r.t the same predicate, here p_1. A dual similarity measure is defined to ensure classification of triples with respect to objects as well as with respect to classes from RDF Schema. The similarity measure is computed based on the taxonomy given in Figure 1. According to the current approach a pairwise similarity measure is computed between the two sets of classes connected through some predicate. For example, $\delta(s_1) \sqcap \delta(s_2) = \langle p_1 : \{C_1, C_2, C_3\}\rangle \sqcap \langle p_1 : \{C_1, C_4, C_5\}\rangle$. Meaning that $\delta(s_1) \sqcap \delta(s_2) = p_1 : \langle lcs(C_1, C_1), lcs(C_1, C_5), \ldots\rangle$. Then, $\delta(s_1) \sqcap \delta(s_2) = p_1 : \{C_1, C_{10}, C_{11}, C_{13}\}$. As we have $C_1 \leqslant C_{11} \leqslant C_{13}$ and $C_{10} \leqslant C_{13}$ then we pick only the specific classes i.e., C_1 and C_{10}. Finally, $\delta(s_1) \sqcap \delta(s_2) = p_1 : \{C_1, C_{10}\}$.

Now let us consider the complete descriptions of the two entities i.e., $\delta(s_1) = \{(p_1 : \{C_1, C_2, C_3\}), (p_2 : \{D_2\})\}$ and $\delta(s_2) = \{(p_1 : \{C_1, C_4, C_5\}), (p_2 : \{D_3\})\}$. Then, $\delta(s_1) \sqcap \delta(s_2) = \{(p_1 : \{C_1, C_2, C_3\}), (p_2 : \{D_2\})\} \sqcap \{(p_1 : \{C_1, C_4, C_5\}), (p_2 : \{D_3\})\}$. The set of least common subsumers for p_1 will be C_1, C_{10}. The least common subumer between $p_2 : \{D_3\}$ and $p_2 : \{D_2\}$ is $p_2 : \{D_5\}$. Finally, the similarity between these two drugs will be $\{(p_1 : \{C_1, C_{10}\}), (p_2 : \{D_5\})\}$.

The proposed similarity measure fulfills the property of a similarity operator i.e., commutative, associative and idempotent [7]. The number of RDF Schema considered are equal to number of predicates in the entity-description representation of an RDF graph. In the running example, we use two schemas to show that our method can also be easily applied to multiple RDF Schemas which are either independent or parts of the same schema distributed over several resources.

4.3 Building the Semantic Pattern Concept Lattice

For building the semantic pattern concept lattice using the above similarity measure over several RDF Schemas, according to the running example, $\delta(s_1) \sqcap \delta(s_2) = \{(p_1 : \{C_1, C_{10}\}), (p_2 : \{D_5\})\}$. The semantic pattern concept lattice is built according to definition 1. For instance, $\{s_1, s_2\}^{\square} = \{(p_1 : \{C_1, C_{10}\}), (p_2 : \{D_5\})\}$ but $\{(p_1 : \{C_1, C_{10}\}), (p_2 : \{D_5\})\}^{\square} = \{s_1, s_2, s_5\}$. This way we obtain $K\#2$ (see Figure 3) with 3 drugs giving the concept a support of 3. Finally, the pattern concept lattice is built for Table 3 with respect to $\mathcal{T}_1$(Figure 1) and $\mathcal{T}_2$(Figure 2). This pattern concept lattice is shown in Figure 3 and the details of each of the pattern concept is shown in Table 4.

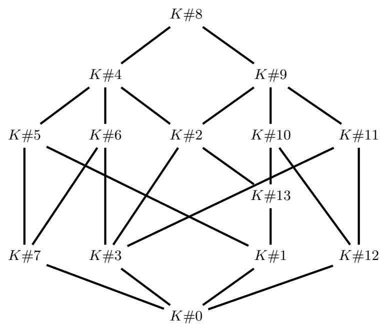


Fig. 3: Pattern Concept lattice for Drugbank

$K\#ID$	**Extent**	**Intent**
$K\#1$	$\{s_1\}$	$(p_1 : \{C_1, C_2, C_3\})$, $(p_2 : \{D_2\})$
$K\#2$	$\{s_1, s_2, s_5\}$	$(p_1 : \{C_1, C_{10}\})$, $(p_2 : \{D_5\})$
$K\#3$	$\{s_2\}$	$(p_1 : \{C_1, C_4, C_5\})$, $(p_2 : \{D_3\})$
$K\#4$	$\{s_1, s_2, s_3, s_5\}$	$(p_1 : \{C_{11}\})$, $(p_2 : \{D_5\})$
$K\#5$	$\{s_1, s_3\}$	$(p_1 : \{C_2\})$, $(p_2 : \{D_5\})$
$K\#6$	$\{s_2, s_3\}$	$(p_1 : \{C_{11}\})$, $(p_2 : \{D_3\})$
$K\#7$	$\{s_3\}$	$(p_1 : \{C_2\})$, $(p_2 : \{D_3\})$
$K\#8$	$\{s_1, s_2, s_3, s_4, s_5\}$	$(p_1 : \{C_{13}\})$, $(p_2 : \{D_8\})$
$K\#9$	$\{s_1, s_2, s_4, s_5\}$	$(p_1 : \{C_{10}\})$, $(p_2 : \{D_8\})$
$K\#10$	$\{s_1, s_4, s_5\}$	$(p_1 : \{C_3\})$, $(p_2 : \{D_8\})$
$K\#11$	$\{s_2, s_4\}$	$(p_1 : \{C_4, C_5\})$, $(p_2 : \{D_8\})$
$K\#12$	$\{s_4\}$	$(p_1 : \{C_3, C_4, C_5\})$, $(p_2 : \{D_9\})$
$K\#13$	$\{s_1, s_5\}$	$(p_1 : \{C_1, C_3\})$, $(p_2 : \{D_2\})$

Table 4: Details of *Pattern Concept lattice* in Figure 3

4.4 Interpreting and Navigating the Pattern Concept Lattice

This pattern concept lattice not only serves as a navigation space but also provides clustering of RDF triples w.r.t. some Schema. Each of the pattern concepts is a cluster of triples. For instance, consider the simplest example in Table 4 i.e., $K\#1$, where there is only one entity in the extent, then this cluster contains the triples $\{(s_1, p_1, C_1), (s_1, p_2, D_2), \dots\}$. Here it can be seen that all the classes are connected to single subject S through two predicates. This space can further be navigated to provide the required information to the user without the technical background of Semantic Web. Let us consider, if user is looking for drugs causing side effect C_{10}, the located pattern concept will be $K\#9$ which contains all the drugs causing some allergic conditions. Afterwards, if the user is looking for some very specific allergic condition such as C_3 then he can simply navigate downwards to obtain more specific concept i.e., $K\#10$ which contains all the drugs which cause the side effect C_3. Let us consider the scenario described in section 2, here the user will be looking for cardiovascular agents (D_5) which do not cause any allergic conditions. In this case first the concept containing all the cardiovascular agents are located i.e., $K\#4$ afterwards the lattice is further navigated downwards to obtain the drugs causing C_{10} (allergic conditions) i.e., $K\#2$. Now the drugs which are in $K\#4$ and not in $K\#2$ are the CVAs which do not cause allergic conditions.

5 SPARQL Queries over Pattern Concept Lattice

For the user with the basic knowledge of SPARQL queries, this *navigation space* can be easily accessed by posing simpler queries. This sub-lattice gives additional information related to the query (detailed below). The obtained pattern concept lattice keeps the materialized RDF Schema meaning that it does not require the query to perform reasoning. It also allows for accessing several resources from one platform. This *navigation space* allows queries which are subject, predicate or object bound. In this section we describe simple as well as complex SPARQL queries that can be posed over the this navigation space. Simple queries refer to the queries with simple Basic Graph Patterns such a single triple pattern in the `WHERE` clause of the SPARQL query, on the other hand, complex queries allow joins and negations.

5.1 Simple Queries

Simple queries include subject, predicate or object bound queries. *Subject-bound queries* refer to the queries where the value of the subject in an RDF triple $\langle s, p, o\rangle$ is known while the values of predicate, object and classes of objects are to be retrieved. Let a SPARQL query be `SELECT ?p ?o WHERE` $\{s_i$ `?p ?o`$\}$, where s_i is a known subject and a pattern concept be (A, d) such that $s_i \in A$. The BGP in this query is (s_i `?p ?o`), this BGP is matched against the RDF graph present in the pattern concept lattice. The answer of the query will be the object concept $\{s_i\}^{\square} = \{d_i\}$ where $d_i \in d$ where d is the intent of the concept (A, d). Note that this object concept is already computed while building a pattern concept lattice. In other words, all the predicate-object pairs $\{d_i\}$ connected to the subject s_i in the RDF graph $\mathcal{G}$ are returned as a result. For example, if user wants to know complete information about the drug s_1 i.e., its side effect and the category, then `SELECT ?p ?o WHERE` $\{s_1$ `?p ?o`$\}$. It will search for object concept in the pattern concept lattice given in Figure 3. For instance, $s_1^{\square} = (p_1 : \{C_1, C_2, C_3\}, p_2 : \{D_2\})$ $(K\#1)$ and the answer would be $(p_1 : \{C_1, C_2, C_3\}, p_2 : \{D_2\})$. The first part of the answer $(p_1 : \{C_1, C_2, C_3\}$ belongs to SIDER while $p_2 : \{D_2\}$ belongs to DrugBank. However, because of the strong lattice-structure a sub-lattice is retrieved by navigating to more general concepts (upward navigation) present in the pattern concept lattice i.e., all the concepts connected to $K\#1$ i.e., $K\#2, K\#4, K\#5, K\#9, K\#10, K\#13, K\#8$. Here, $K\#1$ is the *focus concept* because it keeps the exact answer to the posed

SPARQL query and the rest of the concepts contain additional information. This way the user not only gets the desired answer but also some additional information about the drug i.e., the classification of side effects from MedDRA and the features it shares with other drugs and with which drugs it shares certain features. Object-bound queries are answered in the same way as described above by retrieving the attribute concept and the linked concepts. However, in this case the additional information will be retrieved by navigating downwards from the focus concept because it is the object-bound query and objects are contained in the intent of the pattern concepts.

5.2 Queries with Joins

Now let us consider a slightly more complex query which includes a join and a class resources to be retrieved. If it is specified that the drugs to be retrieved should be Cardiovascular Agent causing some Allergic Condition. Then the SPARQL query with subject-subject join will be as follows:

`SELECT ?drug WHERE {`
`?drug` p_2 D_5`.`
`?drug` p_1 C_{10} `}`

For obtaining the answer, the BGP in the above query is matched against the concept which contains both the predicates p_1 and p_2, along with this predicate it keeps the classes of objects `CardiovascularAgents (`D_5`)` connected to predicate p_2 as well as `AllergicConditions (`C_{10}`)` connected to the predicate p_1. The answer will be $\{s_1, s_2, s_5\}$ i.e., $K\#2$, which will be the focus concept. However, to obtain more specific information regarding these classes of drug categories and the side effects will be returned by navigating downwards in the concept lattice. Finally, a sub-lattice containing $K\#2, K\#3, K\#13, K\#1$ is returned as an answer to the above query. In this case, the user will have an additional information regarding more specific categories related to Cardiovascular Agents causing Allergic Conditions such as according to K#13 drugs $\{s_1, s_5\}$ are D_2 and K#3 $\{s_2\}$ is D_3. Similarly, specific allergic conditions are also retrieved.

5.3 Queries with Filtering Criteria

Now let us move towards the scenario where the doctor is looking for drug replacement for the patient having heart conditions based on its side effects. For replacing the drugs, the system can be queried by locating all the CVAs (this information comes from DrugBank, MeSH in the current navigation space) and then excluding all the CVAs causing Allergic Conditions. The related SPARQL query can be given as follows:

`SELECT ?drug WHERE {`
`?drug` p_2 D_5`.`

```
?drug $p_1$ ?se
FILTER (?se != $C_{10}$) }
```

The condition is given in the filter clause because SPARQL does not support negation operation directly. In order to do so, all the drugs which are CVAs are retrieved by SPARQL and then it filters the drugs which do not cause allergic conditions. Same is the case with the lattice operation where concept K#4 is selected because it contains all the cardiovascular agents and then a more specific concept is obtained containing the CVAs causing allergic conditions i.e., $K\#2$ and minus operator is applied to perform the desired filtering. The answer obtained is s_3.

In a sense, the navigation space follows the query mechanism provided by Google where a query is posed by the user and this query is mapped to a pre-existing cluster of documents. In the same way, when a query is posed by the user, our query mechanism maps the query to a cluster of RDF triples, where each element in an RDF triple keeps the URI of a web-page.

6 Experimentation

The current approach was applied to biomedical data. This section details the experimental evaluation for the creation of the *navigation space* The proposed algorithm was coded in C++ and the experiments was performed using 3GB RAM on Ubuntu version 12.04.

6.1 Drug Search

The datasets containing information about drugs are DrugBank, SIDER, MedDRA and MeSH as described in section 3.1. *DrugBank* keeps detailed information about each of the drugs, their categories and the proteins it targets. The other database is SIDER which keeps the side effects of the drugs contained on the packaging of the drug. The access to these two data sets is provided by University of Mannheim through two different endpoints[13][14].

The schema associated to the side effects of the drug is available on *BioPortal* [12], which is a web portal that provides access to a repository of biomedical ontologies. *BioPortal* is developed by National Center for Biomedical Ontology (NCBO) [10]. The Medical Dictionary for Regulatory Activities (MedDRA) Terminology is the international medical terminology. During this experiment we will enrich side effects with schema level information using MedDRA terminology. In case of the drug categories MeSH vocabulary thesaurus was taken into account. MeSH (Medical Subject Headings) is a controlled vocabulary thesaurus. The drug categories from DrugBank will be enriched with the tree numbers from MeSH vocabulary. The tree numbers arrange the terms from MeSH in a hierarchical manner known as *MeSH Tree Structures*. In the current experiment we

[13] `http://wifo5-04.informatik.uni-mannheim.de/drugbank/snorql/`
[14] `http://wifo5-04.informatik.uni-mannheim.de/sider/snorql/`

used the MeSH vocabulary already present in the form of RDF in Bio2RDF [1, 5], which makes the public databases related to Bioinformatics such Kegg, MeSH, HGNC etc. available in the RDF format.

During this experiment, two subsets of the dataset were considered. Both belonging to two major classes of drugs i.e., Cardiovascular Agents (CVA) and Central Nervous System (CNS).

6.2 Evaluation

In the following, we study the scalability of *Semantic Pattern Structures* over large dataset. Table 5 precises the statistics of the data. Pattern concept lattices over both the chosen data sources was built in 0-25 seconds for the maximum of 31098 triples. Figure 4b shows the variation in the size of the navigation space for both data sets. The navigation space contains a maximum of around 500000 clusters of triples which were generated in 25 seconds. However, there are several ways to reduce these number of concepts. The filtering based on the depth of classes considered in the taxonomy which allows the reduction in the number of clusters while generating the concept lattice and hence causes decrease in the runtime of creating the navigation space. Most of these very general classes are not very interesting for the domain expert. Moreover, there are several measures such as support, stability and lift which allow post-filtering of the navigation space.

Datasets	**No. of Triples**	**No. of Subjects**	**No. of Objects**	**Runtime**
Cardiovascular Agents	31098	145	927	0-22 sec
Central Nervous System	22680	105	1050	0-25 sec

Table 5: Statistics of two datasets and navigation space.

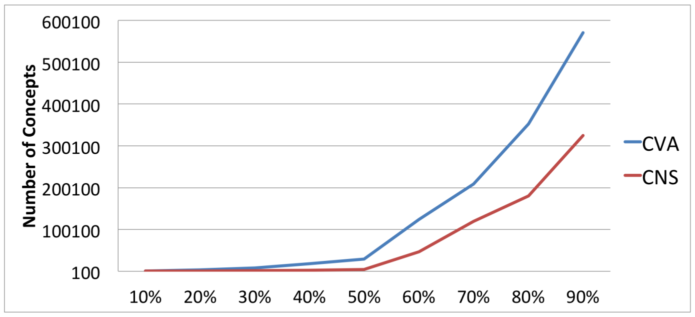


Fig. 4: Size of the Navigation Space for each dataset.

7 Related work

In [6], the author focuses on allowing conceptual navigation to RDF graphs, where each concept is accessed through SPARQL-like queries. However, in our case several RDF graphs are considered and we use the already existing, well-established and well-documented query language, SPARQL. Moreover, [3] introduces ontological pattern structures for enriching raw data with $\mathcal{EL}$ ontologies. But both the approaches consider only one resource at a time, hence not targeting the problem of decentralization. As a contrast, our approach provide navigation space over RDF graphs as well as schema level information from several resources allowing user to access information from one platform. In [4], the authors introduce a new clause `Categorize By` which clusters SPARQL query answers w.r.t to background knowledge present as ontologies. Like our approach, only taxonomy is used for enriching, however, unlike our approach if clusters the answers after obtaining the query answers. As a contrast, our approach provides classification of the RDF triples as well as RDF Schema before hand. Afterwards, SPARQL queries are mapped to the existing clusters and the answer is shown to the user. In such a case, the query response time is faster than the `Categorize By` clause. Moreover, as it provides clustering over answers only, it lacks the capability to provide user with additional information.

8 Conclusion and Future Work

This paper proposes a new approach for navigating semantic web data and targets the capabilities of FCA to deal with RDF data. It provides navigational and search capabilities over RDF triples as well as RDF Schema distributed over several resources. This paper proposes a new similarity measure for pattern structures to deal with RDF data as well as RDF Schema simultaneously, termed as Semantic Pattern Structures. The pattern concepts in the concept lattice are considered as clusters of of RDF triples which can then be navigated and queried by the user.

References

1. François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5):706–716, 2008.
2. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
3. Adrien Coulet, Florent Domenach, Mehdi Kaytoue, and Amedeo Napoli. Using pattern structures for analyzing ontology-based annotations of biomedical data. In *11th International Conference on Formal Concept Analysis,*, 2013.
4. Claudia d'Amato, Nicola Fanizzi, and Agnieszka Lawrynowicz. Categorize by: Deductive aggregation of semantic web query results. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2010.

5. Michel Dumontier, Alison Callahan, Jose Cruz-Toledo, Peter Ansell, Vincent Emonet, François Belleau, and Arnaud Droit. Bio2rdf release 3: A larger, more connected network of linked data for the life sciences. In *Posters & Demonstrations Track ISWC.*, 2014.
6. Sébastien Ferré. Conceptual navigation in RDF graphs with sparql-like queries. In *Formal Concept Analysis, 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings*, pages 193–208, 2010.
7. Bernhard Ganter and Sergei O. Kuznetsov. Pattern structures and their projections. In Harry S. Delugach and Gerd Stumme, editors, *ICCS*, volume 2120 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2001.
8. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin/Heidelberg, 1999.
9. Ralf Küsters and Ralf Molitor. Computing least common subsumers in ALEN. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, pages 219–224, 2001.
10. Mark A. Musen, Natalya Fridman Noy, Nigam H. Shah, Patricia L. Whetzel, Christopher G. Chute, Margaret-Anne D. Storey, and Barry Smith. The national center for biomedical ontology. *JAMIA*, 19(2):190–195, 2012.
11. York Sure, Stephan Bloehdorn, Peter Haase, Jens Hartmann, and Daniel Oberle. The swrc ontology - semantic web for research communities. In *EPIA*, volume 3808 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 2005.
12. Patricia L. Whetzel, Natalya Fridman Noy, Nigam H. Shah, Paul R. Alexander, Csongor Nyulas, Tania Tudorache, and Mark A. Musen. Bioportal: enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. *Nucleic Acids Research*, 39(Web-Server-Issue):541–545, 2011.

Subdirect Decomposition of Contexts into Subdirectly Irreducible Factors

Jean-François Viaud[1], Karell Bertet[1], Christophe Demko[1], Rokia Missaoui[2]

[1] Laboratory L3i, University of La Rochelle, France
{jviaud, kbertet, cdemko}@univ-lr.fr
[2] University of Québec in Outaouais, Canada
rokia.missaoui@uqo.ca

Abstract. The size of a concept lattice may increase exponentially with the size of the context. When the number of nodes is too large, it becomes very difficult to generate and study such a concept lattice. A way to avoid this problem is to break down the lattice into small parts. In the subdirect decomposition, the small parts are factor lattices which are meaningful in the Formal Concept Analysis (FCA) setting.
In this paper a walkthrough from a finite reduced context to its subdirect decomposition into subdirectly irreducible subcontexts and factors is given. The decomposition can be reached using three different points of view, namely factor lattices, arrow relations and compatible subcontexts. The approach is mainly algebraic since it is based on abstract lattice theory, except for the last point which is inherited from FCA. We also propose a polynomial algorithm to generate the decomposition of an initial context into subcontexts. Such a procedure can be extended to conduct an interactive exploration and mining of large contexts, including the generation of few concepts and their neighborhood.

Keywords: concept lattice, congruence relation, factor lattice, arrow relation, arrow closed subcontext, compatible subcontext

1 Introduction

During the last decade, the computation capabilities have promoted Formal Concept Analysis (FCA) with new methods based on concept lattices. Though they are exponential in space/time in worst case, concept lattices of a reasonable size enable an intuitive representation of data organized by a context that links objects to attributes through a binary relation. Methods based on concept lattices have been developed in various domains such as knowledge discovery and representation, database management and information retrieval where some relevant concepts, *i.e.* possible correspondences between objects and attributes are considered either as classifiers, clusters or representative object/attribute subsets.

With the increasing size of data, a set of methods have been proposed in order to either generate a subset (rather than the whole set) of concepts and

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors. Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

their neighborhood (e.g. successors and predecessors) in an online and interactive way [8, 20] or better display lattices using nested line diagrams [13]. Such approaches become inefficient when contexts are huge. However, the main idea of lattice/context decomposition into smaller ones is still relevant when the classification property of the initial lattice is maintained. Many lattice decompositions have been defined and studied, both from algebraic [6, 16] and FCA points of view [13, 12]. We can cite Unique Factorisation Theorem [16], matrix decomposition [2], Atlas decomposition [13], subtensorial decomposition [13], doubling convex construction [5, 17, 14, 3] and subdirect decomposition. The latter has been widely studied many years ago, in the field of universal algebra [6, 11], and even in FCA [21–24, 12]. To the best of our knowledge, there is no new development or novel algorithms for subdirect decomposition of contexts.

In this paper we investigate the subdirect decomposition of a concept lattice as a first step towards an interactive exploration and mining of large contexts. The subdirect decomposition of a lattice L into factor lattices $(L_i)_{i\in\{1,\dots,n\}}$, denoted by $L \hookrightarrow L_1 \times \cdots \times L_n$, is defined by two properties (see important results in [13]): (i) L is a sublattice of the direct product $L_1 \times \cdots \times L_n$, and (ii) each projection of L onto a factor is surjective. The first property establishes that each factor lattice is the concept lattice of an arrow-closed subcontext, *i.e.* closed according to the arrow relation between objects and attributes. This means that the decomposition can be obtained by computing specific subcontexts. The second property states that there is an equivalence between arrow-closed subcontexts and congruence relations of L, *i.e.*, an equivalence relation whose equivalence classes form a lattice with elements closed by the meet and join operations. This means that the concepts of L can be retrieved from the factor lattices, and the classification property of L is maintained since each equivalence relation forms a partition of the elements. The last result establishes an equivalence between arrow-closed subcontexts and compatible subcontexts, *i.e.* subcontexts such that each concept corresponds to a concept of the initial lattice. This result gives a way to compute the morphism from L into the direct product, and thus to retrieve the concepts of L from the factor lattices. In this paper, we deduce from these results strong links between the following notions that have not been used yet together as far as we know:

– Factors of a subdirect decomposition,
– Congruence relations,
– Arrow-closed subcontexts and
– Compatible subcontexts.

As suggested in [13], the contexts of the factors of a particular subdirect decomposition, namely the subdirectly irreducible subcontexts, can be obtained by a polynomial processing of each row/object of the initial context. Therefore, the subdirect decomposition of a lattice can be extended to a subdirect decomposition of its reduced context into subdirect and irreducible subcontexts.

In this paper, we propose a subdirect and polynomial decomposition of a context into subcontexts by extending the subdirect decomposition of a lattice into factor lattices. This decomposition leads to data storage saving of large contexts. Indeed, the generation of the whole set of factor lattices can be avoided by providing an interactive generation of a few (but not all) concepts and their neighborhood from large contexts. Moreover, a focus on a specific factor lattice can be proposed to the user by generating, partially or entirely, the concept lattice and/or a basis of implications.

There are at least two reasons for studying this case of pattern management. The first one comes from the fact that users tend to be overwhelmed by the knowledge extracted from data, even when the input is relatively small. The second reason is that the community of FCA has made progress in lattice construction and exploration, and hence existing solutions can be adapted and enriched to only target useful patterns (*i.e.* pieces of knowledge).

This paper is organized as follows. Section 2 introduces the subdirect decomposition and the three different points of view, namely factor lattices, arrow relations and compatible subcontexts. Section 3 contains the main contribution of this paper about the subdirect decomposition and the proposed algorithms. A preliminary empirical study is presented in Section 4 while Section 5 presents future work.

2 Structural framework

Throughout this paper all sets (and thus lattices) are considered to be finite.

2.1 Lattices and Formal Concept Analysis

Algebraic lattice Let us first recall that a *lattice* $(L, \leq)$ is an ordered set in which every pair (x, y) of elements has a least upper bound, called *join* $x \vee y$, and a greatest lower bound, called *meet* $x \wedge y$. As we are only considering finite structures, every subset $A \subset L$ has a join and meet (e. g. finite lattices are complete).

Concept or Galois Lattice A (formal) *context* (O, A, R) is defined by a set O of objects, a set A of attributes, and a binary relation $R \subset O \times A$, between O and A. Two operators are derived:

- for each subset $X \subset O$, we define $X' = \{m \in A, j \; R \; m \; \forall j \in X\}$ and dually,
- for each subset $Y \subset A$, we define $Y' = \{j \in O, j \; R \; m \; \forall m \in Y\}$.

A (formal) *concept* represents a maximal objects-attributes correspondence by a pair (X, Y) such that $X' = Y$ and $Y' = X$. The sets X and Y are respectively called *extent* and *intent* of the concept. The set of concepts derived from a context is ordered as follows:

$$(X_1, Y_1) \leq (X_2, Y_2) \Longleftrightarrow X_1 \subseteq X_2 \Longleftrightarrow Y_2 \subseteq Y_1$$

The whole set of formal concepts together with this order relation form a complete lattice, called the *concept lattice* of the context (O, A, R).

Different formal contexts can provide isomorphic concept lattices, and there exists a unique one, named the *reduced context*, defined by the two sets O and A of the smallest size.

This particular context is introduced by means of special concepts or elements of the lattice L, namely irreducible elements.

An element $j \in L$ is *join-irreducible* if it is not a least upper bound of a subset not containing it. The set of join irreducible elements is noted J_L. *Meet-irreducible* elements are defined dually and their set is M_L. As a direct consequence, an element $j \in L$ is join-irreducible if and only if it has only one immediate predecessor denoted j^-. Dually, an element $m \in L$ is meet-irreducible if and only if it has only one immediate successor denoted m^+.

In Figure 1, join-irreducible nodes are labelled with a number and meet-irreducible nodes are labelled with a letter.

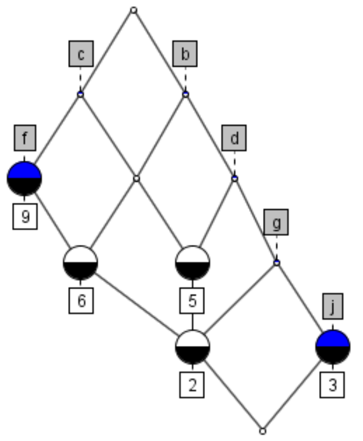


Fig. 1. A lattice with its irreducible nodes

Fundamental Bijection A fundamental result [1] establishes that any lattice $(L, \leq)$ is isomorphic to the concept lattice of the context $(J_L, M_L, \leq)$, where J_L and M_L are the join and meet irreducible concepts of L, respectively. Moreover, this context is a reduced one.

As a direct consequence, there is a bijection between lattices and reduced contexts where objects of the context are associated with join-irreducible concepts of the lattice, and attributes are associated with meet-irreducible concepts.

Figure 2 shows the reduced context of the lattice in Figure 1.

	b	c	d	f	g	j
2	x	x	x	x	x	
3	x		x		x	x
5	x	x	x			
6	x	x		x		
9		x		x		

Fig. 2. The reduced context of the lattice in Figure 1

2.2 Compatible and Arrow-closed Subcontexts

This section is dedicated to the equivalence between compatible and arrow-closed subcontexts.

Compatible subcontexts A *subcontext* of a formal context (O, A, R) is a triple $(J, M, R \cap J \times M)$ such that $J \subset O$ and $M \subset A$. A subcontext $(J, M, R \cap J \times M)$ of (O, A, R) is *compatible* if for each concept (H, N) of (O, A, R), $(J \cap H, M \cap N)$ is a concept of $(J, M, R \cap J \times M)$.

Arrow relations The arrow-closed subcontexts involved in the equivalence are based on the arrow relations between join and meet irreducible concepts of a lattice. Consider the reduced context $(J_L, M_L, \leq)$ of a lattice $(L, \leq)$. Arrow relations [4, 15] form a partition of the relation $\not\leq$ (defined by not having $x \leq y$) by considering the immediate predecessor j^- of a join-irreducible j, and the unique immediate successor m^+ of a meet-irreducible m:

- $j \updownarrow m$ if $j \not\leq m$, $j \leq m^+$ and $j^- \leq m$.
- $j \uparrow m$ if $j \not\leq m$, $j \leq m^+$ and $j^- \not\leq m$.
- $j \downarrow m$ if $j \not\leq m$, $j \not\leq m^+$ and $j^- \leq m$.
- $j \circ m$ if $j \not\leq m$, $j \not\leq m^+$ and $j^- \not\leq m$.

In Figure 3, the reduced context of Figure 2 is enriched with the four relations $\updownarrow$, $\uparrow$, $\downarrow$, and $\circ$ in the empty cells that both correspond to the case where $j \not\leq m$:

	b	c	d	f	g	j
2	$\times$	$\times$	$\times$	$\times$	$\times$	$\updownarrow$
3	$\times$	$\updownarrow$	$\times$	$\downarrow$	$\times$	$\times$
5	$\times$	$\times$	$\times$	$\updownarrow$	$\updownarrow$	$\circ$
6	$\times$	$\times$	$\updownarrow$	$\times$	$\downarrow$	$\circ$
9	$\updownarrow$	$\times$	$\circ$	$\times$	$\circ$	$\circ$

Fig. 3. Arrow relation

As an illustration, let $j = 5$ and $m = f$ be join-irreducible and meet-irreducible nodes respectively (see Figure 1). Then, $j^- = 2$ and $m^+ = c$. The relation $5 \updownarrow f$ holds since $5 \not\leq f$, $5 \leq c$ and $2 \leq f$.

Arrow-closed subcontext A subcontext $(J, M, R \cap J \times M)$ of a context (O, A, R) is an *arrow-closed subcontext* when the following conditions are met:

- If $j \uparrow m$ and $j \in J$ then $m \in M$
- If $j \downarrow m$ and $m \in M$ then $j \in J$

As an example, the first subcontext of Figure 4 is an arrow-closed subcontext of the reduced context of Figure 3 whereas the second one is not, due to the down-arrow $6 \downarrow g$.

	c	d	f	g
3		x		x
5	x	x		
6	x		x	

	c	d	f	g
3		x		x
5	x	x		

Fig. 4. Arrow-closed and non-arrow-closed subcontexts of the context in Figure 3

Equivalence theorem First let us introduce the first equivalence we need in this paper, whose proof can be found in [13]:

Theorem 1. *Let $(J, M, R \cap J \times M)$ be a subcontext of (O, A, R). The following propositions are equivalent:*

- *The subcontext $(J, M, R \cap J \times M)$ is a compatible one.*
- *The subcontext $(J, M, R \cap J \times M)$ is an arrow-closed one.*

2.3 Congruence Relations and Factor Lattices

In this section, we introduce the equivalence between congruence relations and arrow-closed subcontexts.

Quotient An *equivalence relation* is a binary relation R over a set E which is reflexive, symmetric, and transitive. An equivalence class of $x \in E$ is:

$$x_R = \{y \in E \,|\, xRy\}$$

The set of equivalence classes, called the *quotient* set E/R, is:

$$E/R = \{x_R \,|\, x \in E\}$$

Factor lattice A congruence relation Θ on a lattice L is an equivalence relation such that:

$$x_1 \Theta y_1 \text{ and } x_2 \Theta y_2 \Longrightarrow x_1 \wedge x_2 \Theta y_1 \wedge y_2 \text{ and } x_1 \vee x_2 \Theta y_1 \vee y_2$$

The quotient L/Θ verifies the following statement:

$$x_\Theta \leq y_\Theta \Longleftrightarrow x\Theta(x \wedge y) \Longleftrightarrow (x \vee y)\Theta y$$

With such an order, L/Θ is a lattice, called factor lattice. A standard theorem from algebra, whose proof is omitted, states that:

Theorem 2. *The projection $L \rightarrow L/\Theta$ is a lattice morphism onto.*

The second equivalence theorem We are now able to formulate the second equivalence whose proof can be found in [13]:

Theorem 3. *Given a lattice L, the set of congruence relations on L corresponds bijectively with the set of arrow-closed subcontexts of the reduced context of L.*

Congruence relations will be computed with this theorem. However, other algorithms exist [9, 10].

2.4 Subdirect decompositions

In this section, we introduce the equivalence between subdirect decompositions and sets of arrow-closed subcontexts.

Subdirect product

Definition 1. *A subdirect product is a sublattice of a direct product $L_1 \times \cdots \times L_n$ of lattices $L_i, i \in \{1, \ldots, n\}$ such that each projection onto a factor is surjective. The lattices $L_i, i \in \{1, \ldots, n\}$ are the factor lattices. A subdirect decomposition of a lattice L is an isomorphism between L and a subdirect product which can be denoted as:*

$$L \hookrightarrow L_1 \times \cdots \times L_n \twoheadrightarrow L_i$$

The third equivalence theorem The third and most important equivalence whose proof can be found in [13], makes a connection with sets of arrows-closed subcontexts when they cover the initial context:

Proposition 1. *Given a reduced context (O, A, R), then the subdirect decompositions of its concept lattice L correspond bijectively to the families of arrow-closed subcontexts $(J_j, M_j, R \cap J_j \times M_j)$ with $O = \cup J_j$ and $A = \cup M_j$.*

3 Our contribution

3.1 Main Result

From the three previous equivalences found in [13], we deduce the following one:

Corollary 1. *Given a lattice L and its reduced context (O, A, R), we have an equivalence between:*

1. *The set of arrow-closed subcontexts of (O, A, R),*
2. *The set of compatible subcontexts of (O, A, R),*
3. *The set of congruence relations of L and their factor lattices.*

Corollary 2. *Given a lattice L and its reduced context (O, A, R), we have an equivalence between:*

1. *The families of arrow-closed subcontexts of (O, A, R) covering O and A,*
2. *The families of compatible subcontexts of (O, A, R) covering O and A,*
3. *The families $(\theta_i)_{i \in I}$ of congruence relations of L such that $\cap_{i \in I} \theta_i = \Delta$ with $x \Delta y \iff x = y$.*
4. *The set of subdirect decompositions of L and their factor lattices.*

In the following, we exploit these four notions that, to the best of our knowledge, have not been put together yet.

1. The subdirect decomposition ensures that L is a sublattice of the factor lattice product. Moreover, each projection from L to a factor lattice is surjective.
2. The congruence relations of L indicate that factor lattices correspond to their quotient lattices, and thus preserve partitions via equivalence classes.
3. The compatible subcontexts give a way to compute the morphism from L onto its factors.
4. Arrow-closed subcontexts enable the computation of the reduced context of the factor lattices.

In the following we present the generation of a particular subdirect decomposition and show a possible usage of factor lattices.

3.2 Generation of Subdirectly Irreducible Factors

In this section, we consider subdirect decompositions of a lattice L with its reduced context (O, A, R) as input. From Corollary 2, a subdirect decomposition of a lattice L can be obtained by computing a set of arrow-closed subcontexts of (O, A, R) that have to cover O and A. There are many sets of arrow-closed subcontexts and thus many subdirect decompositions. In particular, the decomposition from a lattice L into L itself is a subdirect decomposition, corresponding to the whole subcontext (O, A, R) which is clearly arrow-closed. A subdirect decomposition algorithm has been proposed in [12]. However, all congruence relations are computed and then only pairs of relations are formed to get a decomposition. As a consequence, potentially multiple decompositions are produced with necessarily two factors.

In this article, we focus on the subdirect decomposition of a context into a possibly large number of small factors, *i.e.* factors that cannot be subdirectly decomposed. A factor lattice L is *subdirectly irreducible* when any subdirect decomposition of L leads to L itself. A nice characterization of subdirectly irreducible lattices can be found in [13]:

Proposition 2. *A lattice L is subdirectly irreducible if and only if its reduced context is one-generated.*

A context (O, A, R) is one-generated if it can be obtained by arrow-closing a context with only one $j \in J$. Thus (O, A, R) is the smallest arrow-closed subcontext containing $j \in J$.

Therefore, we deduce the following result:

Proposition 3. *Let L be a lattice. From L, we can deduce a product lattice $L_1 \times ... \times L_n$ where each lattice L_i is:*

- *the concept lattice of a one-generated subcontext,*
- *subdirectly irreducible,*
- *a factor lattice of the subdirectly irreducible decomposition.*

From this result, we propose an algorithm (Algorithm 1) to compute in polynomial time the contexts of the factor lattices $L_1, \ldots, ...L_n$ of a subdirectly irreducible decomposition, with a reduced context (O, A, R) as input. The one-generated subcontexts for each $j \in J$ are obtained by arrow-closing (Algorithm 2). The subdirectly irreducible decomposition of L can then be obtained by computing the concept lattices of these subcontexts.

One can notice that the closure computed from join-irreducible concepts can also be calculated from meet-irreducible concepts.

Algorithm 1: Subdirect_Decomposition

```
Input: A context $(O, A, R)$
Output: List $\mathcal{L}$ of the contexts $(J_j, M_j, R_j)$ of the subdirectly irreducible
        factor lattices
1 $\mathcal{L} \leftarrow \emptyset$;
2 forall the $j \in O$ do
3     Compute $(J_j, M_j, R_j) = $ Arrow_Closure$((j, \emptyset, \emptyset), (O, A, R))$, the
      one-generated subcontext span by $j$;
4     if $\mathcal{L}$ does not contain any subcontext that covers $(J_j, M_j, R_j)$ then
5         add $(J_j, M_j, R_j)$ to $\mathcal{L}$
6     if $\mathcal{L}$ contains a subcontext covered by $(J_j, M_j, R_j)$ then
7         delete it from $\mathcal{L}$
8     return $\mathcal{L}$;
```

Algorithm 2: Arrow_Closure

```
   Input: A subcontext $(\tilde{J}, \tilde{M}, \tilde{R})$ of a context $(J, M, R)$
   Output: Arrow-closure of $(\tilde{J}, \tilde{M}, \tilde{R})$
1  $J_c = \tilde{J}$; $M_c = \tilde{M}$;
2  $pred_J = 0$; $pred_M = 0$;
3  while $pred_M < card(M_c)$ or $pred_J < card(J_c)$ do
4      $pred_J = \text{card}(J_c)$;
5      $pred_M = \text{card}(M_c)$;
6      forall the $j \in J_c$ do
7          add to $M_c$ all $m \in M$ such that $j \uparrow m$;
8      forall the $m \in M_c$ do
9          add to $J_c$ all $j \in J$ such that $j \downarrow m$;
10     Return $(J_c, M_c, R \cap J_c \times M_c)$
```

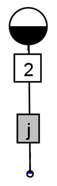
j	Arrow_Closure			Contained in $\mathcal{L}$
	Input	Output		
	$(\tilde{J}, \tilde{M}, \tilde{R})$	J_c	M_c	
2	$(2, \emptyset, \emptyset)$	$\{2\}$	$\{j\}$	×
3	$(3, \emptyset, \emptyset)$	$\{3\}$	$\{c\}$	
5	$(5, \emptyset, \emptyset)$	$\{3, 5, 6\}$	$\{c, d, f, g\}$	×
6	$(6, \emptyset, \emptyset)$	$\{6\}$	$\{d\}$	
9	$(9, \emptyset, \emptyset)$	$\{9\}$	$\{b\}$	×

Fig. 5. Iterations of Algorithm 1 for the reduced context in Figure 2

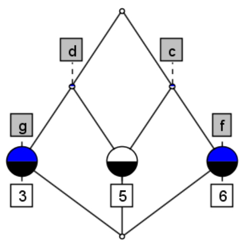
Consider the reduced context in Figure 2. Each iteration of Algorithm 1 is described by Figure 5 for each value of j, the input and output of Algorithm 2, and the three one-generated subcontexts that belong to $\mathcal{L}$ at the end of the process. Therefore we get three factor lattices (see Figure 6).

The first subcontext is the one on the left of Figure 4. The two other ones are: $(\{2\}, \{j\}, \emptyset)$ and $(\{9\}, \{b\}, \emptyset)$. The latter two subcontexts are interesting because:

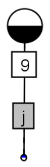
- They show that the initial lattice has parts which are distributive. Indeed, these two subcontexts contain exactly one double arrow in each line and each column.
- They give us a dichotomy: any concept contains either 2 or j ; any concept contains either 9 or b
- In the reduced context, an arrow brings a deeper knowledge than a cross.



(a) $(\{2\}, \{j\}, \emptyset)$



(b) First subcontext in Figure 4



(c) $(\{9\}, \{b\}, \emptyset)$

Fig. 6. The three factor lattices of the decompostion with their subcontext as caption

The context on the left hand-side of Figure 4 is tricky to understand. For the other ones, we have a simple relation $2 \updownarrow j$ or $9 \updownarrow b$, which means that, for instance, 2 and j are some kind of complement or converse.

Figure 7 shows a factor lattice and its corresponding congruence.

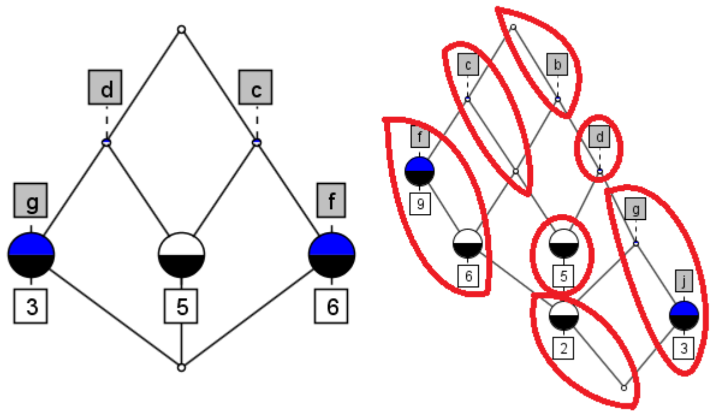


Fig. 7. Factor lattice and congruence

3.3 Onto Morphism and FCA

A subdirect decomposition of a lattice L into factor lattices $L_1, \ldots, L_n$ is relevant since there exists an into morphism from L to the product lattice $L_1 \times \ldots \times L_n$. This morphism is specified by the bijection between compatible subcontexts and congruence relations stated by Corollary 1:

Proposition 4. *Let $(J, M, R \cap J \times M)$ be a compatible subcontext, then the relation $\Theta_{J,M}$ defined by:*

$$(A_1, B_1)\Theta_{J,M}(A_2, B_2) \Longleftrightarrow A_1 \cap J = A_2 \cap J \Longleftrightarrow B_1 \cap M = B_2 \cap M$$

is a congruence relation, and its factor lattice is isomorphic to the concept lattice of the subcontext $(J, M, R \cap J \times M)$.

Algorithm 3 computes this morphism: each concept of L is computed as the product of concepts in each factor, and then marked in the product lattice. From Algorithm 3, we get the large tagged lattice product shown in Figure 8. Obviously, this algorithm is not intended to be used in a real application with large contexts since the product lattice is much more bigger than the original one, while the main goal of the decomposition is to get smaller lattices. We only use this algorithm in the empirical study.

Nevertheless, this morphism can be extended to develop basic FCA processing. Once the subdirectly irreducible decomposition of a reduced context (O, A, R) into the contexts $C_1, \ldots, C_n$ is computed, an interactive exploration and mining process can easily be considered by using the following basic tasks and avoiding the generation of the lattice for the whole context (O, A, R):

- Compute the smallest concept of L that contains a given subset of objects or attributes, and identify its neighborhood
- Compute the smallest concept c_{ij} and its neighborhood in a subset of factors that contain a given collection of objects or attributes. Each factor L_i is a specific view of data.

Algorithm 3: Into_morphism

```
Input: Initial lattice $L$;
Subcontexts $(J_j, M_j, R_j)$;
Product lattice $P = L_1 \times \ldots \times L_n$
Output: Product lattice $P$ with nodes coming from $L$ marked.
1 forall the $c = (A, B) \in L$ do
2     forall the $(J_j, M_j, R_j)$ do
3         Compute $(A \cap J_j, B \cap M_j)$;
4     Mark, in $P$, the product node $\Pi_j(A \cap J_j, B \cap M_j)$;
```

4 Experiments

In this section, we conduct an empirical study in order to better understand the impact of the subdirect decomposition on contexts with different density levels. All the tests were done using the java-lattices library available at:

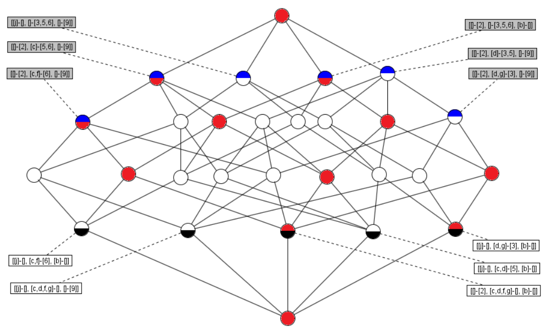
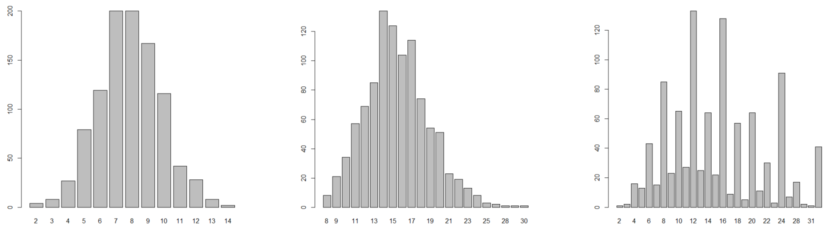

[[j]-[], []-[3,5,6], []-[9]]
[[]-[2], [c]-[5,6], []-[9]]
[[]-[2], [c,f]-[6], []-[9]]
[[]-[2], []-[3,5,6], [b]-[]]
[[]-[2], [d]-[3,5], []-[9]]
[[]-[2], [d,g]-[3], []-[9]]
[[j]-[], [c,f]-[6], [b]-[]]
[[j]-[], [c,d,f,g]-[], []-[9]]
[[j]-[], [d,g]-[3], [b]-[]]
[[j]-[], [c,d]-[5], [b]-[]]
[[]-[2], [c,d,f,g]-[], [b]-[]]

Fig. 8. Ugly tagged lattice product

http://thegalactic.github.io/java-lattices/

A thousand of contexts of 10 observations and 5 attributes were randomly generated. The experiments have been done three times using three density values, namely 20%, 50% and 80%. Figure 9 presents the number of generated lattices according to their size and their density. We can observe two histograms with a nice gaussian shape in the first two cases, but a strange behavior in the last case. However, we roughly observe that the higher the density is, the bigger the lattices are. Therefore, the context density will have an impact on the decomposition process.



(a) Density of 20% (b) Density of 50% (c) Density of 80%

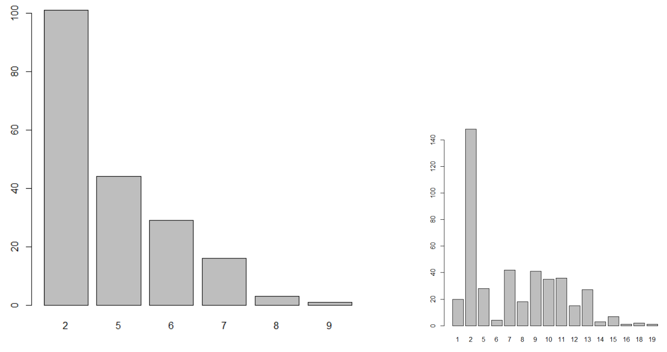
Fig. 9. Number of lattices per size.

The number of generated factors is given in Figure 10. One can observe that this number increases with the density of the initial context since the corresponding lattices have more edges. With a context density of 20%, we get 87.40% irreducible contexts. However, with a density of 80%, only 1.50% of contexts are irreducible and 70% of contexts have at least 4 factors in their decomposition. Thus, lattices are more decomposable when they come from dense contexts.

Factors	Density=20%	Density=50%	Density=80%
1	87.40%	70.50%	1.50%
2	9.70%	21.40%	9.90%
3	2.60%	6.20%	18.70%
4	0.30%	0.50%	23.40%
5		1.40%	28.50%
6			18.00%

Fig. 10. Proportion of the number of factors in the subdirect decomposition of the contexts according to their density

Let us now examine the size of factors with two different density values, namely 20% and 50%. Figure 11 gives the number of cases (initial lattices) according to the number of produced factors. Of course, we observe that smaller lattices give rise to smaller factors. We can also see that in these two density cases, the largest number is obtained for factors of size 2.



(a) Density 20% (b) Density 50%

Fig. 11. Number of cases according to the number of generated factors.

The last part of the tests aims at using contexts with a large density of 80% and computing the ratio between the number of nodes (edges resp.) of the initial lattice and the number of nodes (edges resp.) of the product lattice.

We thus get Figure 12 which shows that the higher is the number of factors, the bigger is the product lattice. Consequently, the set of useless (void) nodes in the product becomes larger as the number of factors increases.

We have also conducted experiments with 40 observations and 15 attributes, and a hundred of contexts were generated using two density values: 20% and 50%. Unfortunately, all were subdirectly irreducible.

Factors	% of nodes	% of edges
1	100%	100%
2	97.90%	97.20%
3	89.20%	84.60%
4	85.00%	77.40%
5	80.60%	71.80%
6	68.00%	57.30%

Fig. 12. Proportion of untagged nodes and edges

5 Conclusion and future work

In this paper, we have presented a polynomial algorithm for the decomposition of a reduced context into subcontexts such that the concept lattice of each subcontext is a subdirectly irreducible factor. This decomposition is a direct consequence inferred from strong links between factors of a subdirect decomposition, congruence relations, arrow-closed subcontexts and compatible subcontexts established in [13].

To further investigate the subdirect decomposition, it would be interesting to conduct large-scale experiments on real world data not only to confirm/nullify the preliminary empirical tests but also to understand the semantics behind the generated irreducible contexts. In particular, attributes covered by several factors interfere in different views of data whose semantics must be interesting to understand. Moreover, it would be useful to allow the user to interactively select a few factors of the decomposition by mixing our approach with the one in [12].

From a theoretical point of view, we think that there are strong links between the implication basis in a quotient lattice and the one from the initial lattice. To the best of our knowledge, this issue has never been addressed and could have significant algorithmic impacts. However, we note that [19] tackle a similar issue in case of a vertical decomposition of a context into subcontexts.

Since the empirical study in [18] show that many real-life contexts are subdirectly irreducible, we plan to (i) identify cases in which a context is necessarily irreducible, and (ii) study, compare and combine other decompositions, in particular the Fratini congruence [7], and the reverse doubling convex construction [5, 17, 14, 3]. Finally, the construction of a lattice from its factor lattices can be done based on the optimization principles behind the relational join operation in databases.

References

1. Barbut, M., Monjardet, B. (eds.): L'ordre et la classification. Algèbre et combinatoire, tome II, Hachette (1970)
2. Belohlavek, R., Vychodil, V.: Discovery of optimal factors in binary data via a novel method of matrix decomposition. Journal of Computer and System Sciences 76(1), 3–20 (Feb 2010)

3. Bertet, K., Caspard, N.: Doubling convec sets in lattices: characterizations and recognition algorithms. Tech. Rep. TR-LACL-2002-08, LACL (Laboratory of Algorithms, Complexity and Logic), University of Paris-Est (Paris 12) (2002)
4. Crawley, P., Dilworth, R.: Algebraic theory of lattices. Prentice Hall, Englewood Cliffs (1973)
5. Day, A.: Congruence normality: The characterization of the doubling class of convex sets. algebra universalis 31(3), 397–406 (1994)
6. Demel, J.: Fast algorithms for finding a subdirect decomposition and interesting congruences of finite algebras. Kybernetika (Prague) 18(2), 121–130 (1982)
7. Duquenne, V.: Lattice drawings and morphisms. In: Formal Concept Analysis, 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings. pp. 88–103 (2010)
8. Ferré, S.: Reconciling Expressivity and Usability in Information Access from File Systems to the Semantic Web. Ph.D. thesis, Univeristy Rennes 1 (2014)
9. Freese, R.: Computing congruence lattices of finite lattices. Proceedings of the American Mathematical Society 125(12), 3457–3463 (1997)
10. Freese, R.: Algorithms in finite, finitely presented and free lattices. Preprint, July 22, 159–178 (1999)
11. Freese, R.: Computing congruences efficiently. Algebra universalis 59(3-4), 337–343 (2008)
12. Funk, P., Lewien, A., Snelting, G.: Algorithms for concept lattice decomposition and their applications. Tech. rep., TU Braunschweig (December 1995)
13. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
14. Geyer, W.: The generalized doubling construction and formal concept analysis. algebra universalis 32(3), 341–367 (1994)
15. Grätzer, G.: General lattice theory. Birkhäuser-Verlag, Basel (1978)
16. Mihók, P., Semanišin, G.: Unique factorization theorem and formal concept analysis. In: Yahia, S., Nguifo, E., Belohlavek, R. (eds.) Concept Lattices and Their Applications, Lecture Notes in Computer Science, vol. 4923, pp. 232–239. Springer Berlin Heidelberg (2008)
17. Nation, J.: Alan day's doubling construction. algebra universalis 34(1), 24–34 (1995)
18. Snelting, G.: Concept lattices in software analysis. In: Formal Concept Analysis, Foundations and Applications. pp. 272–287 (2005)
19. Valtchev, P., Duquenne, V.: On the merge of factor canonical bases. In: International Conference on Formal Concept Analysis ICFCA, pp. 182–198. Springer Berlin Heidelberg (2008)
20. Visani, M., Bertet, K., Ogier, J.M.: Navigala: an Original Symbol Classifier Based on Navigation through a Galois Lattice. International Journal on Pattern Recognition and Artificial Intelligence (IJPRAI) (2011)
21. Wille, R.: Subdirekte produkte und konjunkte summen. Journal für die reine und angewandte Mathematik 0239_0240, 333–338 (1969)
22. Wille, R.: Subdirekte Produkte vollständiger Verbände. J. reine angew. Math. 283/284, 53–70 (1976)
23. Wille, R.: Subdirect decomposition of concept lattices. Algebra Universalis 17, 275–287 (1983)
24. Wille, R.: Subdirect product construction of concept lattices. Discrete Mathematics 63(2-3), 305–313 (1987)

Context-Dependent Deductive and Inductive Reasoning Based on Good Diagnostic Tests

Xenia Naidenova[1] and Vladimir Parkhomenko[2]

[1] Military Medical Academy, Saint-Petersburg, Russia
ksennaidd@gmail.com
[2] Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russia
parhomenko.v@gmail.com

Abstract. A sketch of classification reasoning is given in the paper. The key ideas of the reasoning are ideas of classification and its good approximations based on good diagnostic tests. Such good tests, which are maximally redundant (GMRTs), i.e. their subsets of attributes are closed, are considered. Classification reasoning embraces two interrelated processes: inductive inferring implicative assertions based on GMRTs and using these assertions in deductive steps of reasoning. A peculiarity of our inductive model of classification reasoning is the control and transformation of subcontexts during inductive phase of reasoning by the use of deductive reasoning rules, for example the rules prohibiting some pairs of attributive values.

Keywords: good diagnostic test, classification reasoning, essential attributes, essential objects, implications, subcontexts, deduction, induction, closed sets

1 Introduction

Classification reasoning is a part of commonsense (plausible) reasoning based on two kinds of logical rules extracted from observable datasets (functional and implicative dependencies). The principle concepts of this reasoning are the concepts of classification and its good approximations. We assume that objects are described by a set U of symbolic or numeric attributes. To give a target classification of objects, we use an additional attribute KL not belonging to U. A target attribute partitions a given set of objects into disjoint classes the number of which is equal to the number of values of this attribute. In Tab. 1, we have two classes: KL_+ (positive objects) and KL_- (negative objects).

We concentrate on the case of object classification into two classes and task of concept formation [2,5]. The goal of this task is to describe/classify new objects according to description/classification of existing objects. Inferring good diagnostic tests (GDTs) is the formation of the best descriptions of a given object class KL_+ against the objects not belonging to this class (KL_-).

Let $M = \{\cup \mathrm{dom}(\mathrm{attr}), \mathrm{attr} \in U\}$, where dom(attr) is the set of all values of attr. Let $X \subseteq M$ and G be the set of indices of objects considered (objects

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors. Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

Table 1. Motivating Example of classification

No	Height	Color of Hair	Color of Eyes	KL
1	Low	Blond	Blue	KL_+
2	Low	Brown	Blue	KL_-
3	Tall	Brown	Hazel	KL_-
4	Tall	Blond	Hazel	KL_-
5	Tall	Brown	Blue	KL_-
6	Low	Blond	Hazel	KL_-
7	Tall	Red	Blue	KL_+
8	Tall	Blond	Blue	KL_+

for short), $G = G_+ \cup G_-$, where G_+ and G_- the sets of positive and negative objects, respectively. Denote by $\mathrm{d}(g)$ the description of object $g \in G$. Let $\mathrm{P}(X) = \{g \mid g \in G, X \subseteq \mathrm{d}(g)\}$. We call $\mathrm{P}(X)$ the interpretation of X in the power set 2^G. If $\mathrm{P}(X)$ contains only positive objects and the number of these objects more than 2, then we call X a description of some positive objects and $(\mathrm{P}(X), X)$ a test for positive objects.

Let us define a good test or good description of objects.

Definition 1. *A set $X \subseteq M$ of attribute values is a good description of positive (negative) objects if and only if it is the description of these objects and no such subset $Y \subseteq M$ exists, that $\mathrm{P}(X) \subset \mathrm{P}(Y) \subseteq G_+$.*

It can be seen [15,11,1] that this problem is reduced to searching for causal dependencies in the form $X \rightarrow v, X \subseteq M, v$ is value of attribute KL for all positive (negative) objects.

Sec.2 describes an approach to classification reasoning based on implications. Sec.3 describes the definitions of Good Test Analysis. Sec.4 presents the decomposition of inferring good tests into two kinds of subtasks. An approach to selecting and ordering subcontexts is given in Subsec.5.1. Some situations of reducing the classification context are considered in Subsec.5.2. A set of examples illustrates all the considerations.

2 Classification Reasoning

Classification reasoning based on GDTs embraces two interrelated processes: inductive inferring implicative assertions and using them for pattern recognition problems. We need the following three types of logical rules in order to realize classification reasoning (deductive and inductive):

- INSTANCES or relationships between objects or facts really observed. Instance can be considered as a logical rule with the least degree of generalization. On the other hand, instances can serve as a source of training set of positive and negative examples for inductive inference of generalized rules.

- RULES OF THE FIRST TYPE or logical assertions. These rules describe regular relationships between objects and their properties, between properties of different objects, between objects and their classes. The rules of the first type can be given explicitly by an expert or derived automatically from instances with the help of some learning process. These rules are represented, in the paper, in the form of implications.
- RULES OF THE SECOND TYPE or commonsense reasoning rules with the help of which rules of the first type are used, updated, and inferred from data (instances). The rules of the second type embrace both inductive and deductive reasoning rules.

The rules of the first type can be represented with the use of only one class of logical statements based on implicative dependencies between names. Names are used for designating concepts, things, events, situations, or any evidences. They can be considered as attributes and attributes' values in the formal representations of logical rules.

1. Implication: $a, b, c \rightarrow d$. This rule means that if the values standing on the left side of rule are simultaneously *true*, then the value on the right side of rule is always *true*.
2. Interdiction or forbidden rule (a special case of implication): $a, b, c \rightarrow false$ (*never*). This rule interdicts a combination of values enumerated on the left side of rule. The rule of interdiction can be transformed into several implications such as $a, b \rightarrow$not c; $a, c \rightarrow$not b; $b, c \rightarrow$not a.
3. Compatibility: $a, b, c \rightarrow VA$ where VA is the frequency of occurrence of rule. The compatibility is equivalent to the following implications: $a, b \rightarrow c, VA$; $a, c \rightarrow b, VA$; $b, c \rightarrow b, VA$.
   Generally, the compatibility rule represents the most common combination of values, which is characterized by an insignificant number of exceptions (contrary examples) from regularity.
4. Diagnostic rule: $x, d \rightarrow a$; $x, b \rightarrow$not a; $d, b \rightarrow false$. For example, d and b can be two values of the same attribute. This rule works when the truth of 'x' has been proven and it is necessary to determine whether 'a' is *true* or not. If 'x&d' is *true*, then 'a' is *true*, if 'x&b' is *true*, then 'a' is $false$.
5. Rule of alternatives: a or $b \rightarrow true$ (*always*); $a, b \rightarrow false$. This rule says that a and b cannot be simultaneously *true*, either a or b can be *true* but not both. This rule is a variant of interdiction.

Deductive steps of classification reasoning consist of inferring consequences from some observed facts with the use of implications (i.e. knowledge). For this goal, deductive rules of reasoning are applied the main forms of which are modus ponens, modus tollens, modus ponendo tollens, and modus tollendo ponens. Let x be a set of *true* values of some attributes observed simultaneously. Deductive reasoning rules of the second type are:

1. Using implication: Let r be an implication, left(r) be the left part of r and right(r) be the right part of r. If left(r) $\subseteq x$, then x can be extended by

right(r) : $x \leftarrow x \cup$ right(r). Using implication is based on modus ponens: if A, then B; A; hence B.
2. Using interdiction: Let r be an implication $y \rightarrow$not k. If left$(r) \subseteq x$, then k is the forbidden value for all extensions of x. Using interdiction is based on modus ponendo tollens: either A or B (A, B — alternatives); A; hence not B; either A or B; B; hence not A.
3. Using compatibility: Let $r =$ '$a, b, c \rightarrow k, VA$'. If left$(r) \subseteq x$, then k can be used to extend x along with the calculated value VA for this extension (a special consideration is required for calculating VA).
4. Using diagnostic rules: Let r be a diagnostic rule such as '$x, d \rightarrow a$; $x, b \rightarrow$not a', where 'x' is *true*, and 'a', 'not a' are hypotheses or possible values of some attribute. Using the diagnostic rule is based on modus ponens and modus ponendo tollens. There are several ways for refuting one of the hypotheses:
   - To infer either d or b by using existing knowledge;
   - To involve new known facts (extended context) and/or propositions for inferring (with the use of inductive reasoning rules of the second type) new implications for distinguishing between the hypotheses 'a' and 'not a'; to apply the newly obtained rules;
   - To get the direct answer on whether d or b is *true* by involving an external source of knowledge.
5. Using rule of alternatives: Let 'a' and 'b' be two alternatives (mutually exclusive) hypotheses about the value of some attribute, and the truth of one of hypotheses has been established, then the second hypothesis is rejected. Using the rule is based on modus tollendo ponens: either A or B (A, B — alternatives); not A; hence B; either A or B; not B; hence A.
   We can call the rules listed above the rules of "forward inference". However, we have in natural classification reasoning so called "backward inference".
6. Generating hypothesis or abduction rule. Let r be an implication $y \rightarrow k$. We have "if k is *true*, then y may be *true*".
7. Using modus tollens. Let r be an implication $y \rightarrow k$. If 'not k' is inferred, then 'not y' is also inferred.

When applied, the above given rules generate the reasoning, which is not demonstrative. The purpose of it is to infer all possible hypotheses of the value of some objective attribute. It is essential that hypotheses do not contradict with knowledge (first-type rules) and the observable real situation, where the reasoning takes place (contextual knowledge). Inference of hypotheses is reduced to constructing all intrinsically consistent extensions of a set of values in which the number of involved attributes is maximum possible (there are no prohibited pairs of values in such extensions). All hypotheses have different admissibility determined by the quantity and "quality" of compatibility rules involved in reasoning.

A very simple structure of knowledge base (KB) and an example of applying the rules of second type for a pattern recognition problem can be found in [19]. The process of deductive reasoning can require inferring new rules of the first type from data when i) the result of reasoning contains several hypotheses and it is impossible to choose one and only one of them (uncertainty), and ii) it is impossible to infer any hypothesis.

Inductive steps of classification reasoning consist in using already known facts, observations and experimental results for inferring implications and their correcting. The main forms of induction have been formulated by J. Stuart Mill [12]. The formalization of these forms of induction has been offered by V.K. Finn [6]. For inferring GDTs, the Combined Similarity and Difference Method of the Mill are used and realized by means of inductive inference rules of the second type proposed in [13].

Deductive rules of the second type described above also work for inductive inferring implications since implications obtained are involved immediately to reduce the search space. In the paper, we shall illustrate using the forbidden and diagnostic rules of the second type for GDTs inferring.

3 Key Notions of Good Test Analysis

Assume that $G = \overline{1, N}$ be the set of objects and $M = \{m_1, m_2, \ldots, m_j, \ldots m_m\}$ be the set of values. The object descriptions are represented by rows of a table, columns of which are associated with the attributes taking their values in M (see, please, Tab.1).

Denote by $\{B_g |\, B_g \subseteq M, g \in G\}$ the description of an object. The Galois connection [21] between the ordered sets $(2^G, \subseteq)$ and $(2^M, \subseteq)$, i.e. $2^G \rightarrow 2^M$ and $2^M \rightarrow 2^G$, is defined by the following mappings called derivation operators (see although different notations in [16,9]): for $A \subseteq G$ and $B \subseteq M$, $A' = \mathrm{val}(A) = \{\text{intersection of all } B_g |\, B_g \subseteq M, g \in A\}$ and $B' = \mathrm{obj}(B) = \{g |\, g \in G, B \subseteq B_g\}$. We consider $\mathrm{obj}(B) = \{\text{intersection of all obj}(m) |\, \mathrm{obj}(m) \subseteq G, m \in B\}$. In the Formal Concept Analysis, for $g \in G$ and $m \in M$, $\{g\}'$ or simply g' is called object intent, and $\{m\}'$ or simply m' is called attribute extent [7].

There are two closure operators [16]: generalization_of$(B) = B'' = \mathrm{val}(\mathrm{obj}(B))$ and generalization_of$(A) = A'' = \mathrm{obj}(\mathrm{val}(A))$. A set A is closed if $A = \mathrm{obj}(\mathrm{val}(A))$ and a set B is closed if $B = \mathrm{val}(\mathrm{obj}(B))$. If $A' = B$ & $B' = A$, then a pair (A, B) is called a formal concept, subsets A and B of which are called concept extent and intent, respectively.

A triple (G, M, I), where I is a binary relation $G \times M$ is called a formal context $\mathbb{K}$. Let $\mathbb{K}_\epsilon := (G_\epsilon, M, I_\epsilon)$ and $I_\epsilon := I \cap (G_\epsilon \times M)$, where $\epsilon \in \{+, -\}$ (one can add the value τ if there is an undefined object) [8]. $\mathbb{K}_\pm := (G_+ \cup G_-, M \cup \{\mathrm{KL}\}, I_+ \cup I_- \cup (G_\pm \times \{\mathrm{KL}\}))$.

Definition 2. *A Diagnostic (classification) Test (DT) for G_+ is a pair (A, B) such that $B \subseteq M$, $A = \mathrm{obj}(B) \neq \emptyset$, $A \subseteq G_+$, and $B \not\subseteq \mathrm{val}(g)$, $\forall g \in G_-$. Equivalently,* $\mathrm{obj}(B) \cap G_- = \emptyset$.

Definition 3. *A DT (A, B) for G_+ is maximally redundant if* $\mathrm{obj}(B \cup m) \subset A$ *for all $m \notin B$ and $m \in M$.*

Definition 4. *A DT (A, B) for G_+ is irredundant if any narrowing $B_* = B \setminus m$, $m \in B$ implies that $(obj(B_*), B_*))$ is not a test for G_+.*

Definition 5. *A DT (A, B) for G_+ is good if and only if any extension $A_* = A \cup i$, $i \notin A$, $i \in G_+$ implies that $(A_*, val(A_*))$ is not a test for G_+.*

If a good test (A, B) for G_+ is maximally redundant (GMRT), then any extension $B_* = B \cup m$, $m \notin B$, $m \in M$ implies that $(\mathrm{obj}(B_*), B_*)$ is not a good test for G_+. The relation GMRTs to the Formal Concept Analysis is considered in [17].

Any object description $d(g), g \in G$ in a given classification context is a maximally redundant set of values because $\forall m \notin d(g)$, $m \in M, \mathrm{obj}(d(g) \cup m)$ is equal to $\emptyset$.

In Tab.1, $((1, 8)$, Blond Blue$)$ is a GMRT for KL_+, $((4, 6)$, Blond Hazel$)$ is a maximally redundant test for KL_- but not a good one, and $((3, 4, 6)$, Hazel$)$ is a good maximally redundant and, simultaneously, good irredundant test for KL_-.

4 Decomposition of Inferring GMRTs into Subtasks

There are two kinds of subtasks for inferring GMRTs:

1. given a set of values B, where $B \subseteq M$, $\mathrm{obj}(B) \neq \emptyset$, $B \not\subseteq d(g), \forall g \in G_-$, find all GMRTs $(\mathrm{obj}(B_*), B_*)$ such that $B_* \subset B$;
2. given a non-empty set of values $X \subseteq M$ such that $(\mathrm{obj}(X), X)$ is not a test for positive objects, find all GMRTs $(\mathrm{obj}(Y), Y)$ such that $X \subset Y$.

For solving these subtasks we need to form subcontexts of a given classification context $\mathbb{K}_\pm$. In the first case, we have to include in subcontext all positive objects, the descriptions of which intersect B. The subtask is useful to find all GMRTs intents of which are contained in the description $d(g)$ of an object g. This subtask is considered in [5] for fast incremental concept formation. The following notions of object and value projections are developed to form subcontexts [18].

Definition 6. *The projection $\mathrm{proj}(d(g)), g \in G_+$ on the set D_+, where D_+ is the set of descriptions of all positive objects, is denoted by $\{z | \, z = d(g) \cap d(g_*) \neq \emptyset, d(g_*) \in D_+ \; \& \; (\mathrm{obj}(z), z)$ is a test for $G_+\}$. Note that $d(g) \in \mathrm{proj}(d(g))$.*

Definition 7. *The value projection on a given set D_+ $\mathrm{proj}(B)$ is $\{d(g) \mid B \subseteq d(g),\ d(g) \in D_+\}$ or, equivalently, $\mathrm{proj}(B) = \{d(g) \mid g \in (\mathrm{obj}(B) \cap G_+)\}$.*

The projections define the methods to construct two kinds of subcontexts of $\mathbb{K}_\pm$. The following theorem gives the foundation of reducing subcontexts [14].

Theorem 1. *Let $X \subseteq M$, $(\mathrm{obj}(X), X)$ be a maximally redundant test for positive objects and $\mathrm{obj}(m) \subseteq \mathrm{obj}(X)$, $m \in M$. Then m can not belong to any GMRT for positive objects different from $(obj(X), X)$.*

Let splus(m) be obj$(m) \cap G_+$ (obj$(m) \cap G_-$) and SPLUS=$\{$splus$(m), m \in M\}$. Consider an example of reducing subcontext in Tab. 1.

For values Hazel, Brown, Tall, Blue, Blond, and Low, SPLUS=$\{$obj$(m) \cap G_-\}$ $= \{\{3,4,6\},\{2,3,5\},\{3,4,5\},\{2,5\},\{4,6\},\{2,6\}\}$.

We have val(obj(Hazel)) = Hazel, hence ((3,4,6), Hazel) is a test for G_-. Then value Blond can be deleted from consideration, because splus(Blond) $\subset$ splus(Hazel).

It is essential that the projection is a subset of object descriptions defined on a certain restricted subset $t_* \subseteq M$ of values. Let s_* be the subset of objects, descriptions of which produce the projection. In the projection, splus(m) = obj$(m) \cap s_*$, $m \in t_*$.

Let STGOOD be the partially ordered set of elements s such that $(s, \text{val}(s))$ is a good test for D_+.

Forming the Set STGOOD. The important part of our basic recursive algorithm is how to form the set STGOOD. Let $L = (2^S, \leq)$ be the set lattice [22]. The ordering determined in the set lattice coincides with the set-theoretical inclusion. Subset s_1 is absorbed by subset s_2, i.e. $s_1 \leq s_2$, if and only if $s_1 \subseteq s_2$. Under formation of STGOOD, subset $s \subseteq G_+$ is stored in STGOOD if and only if it is not absorbed by any element of this set. It is necessary also to delete from STGOOD all the elements that are absorbed by s if s is stored in STGOOD. Thus, when the algorithm is over, the set STGOOD contains all the collections of objects that are the extents of GMRTs and only such collections. The set TGOOD of all the GMRTs is obtained as follows: TGOOD = $\{$tg$|$ tg = $(s, \text{val}(s)),\ s \in$ STGOOD$\}$.

The basic recursive procedure for solving any kind of subtasks has been described in [20].

5 Selecting and Ordering Subcontexts and Inferring GMRTs

Algorithms of GMRTs inferring are constructed by the rules of selecting and ordering subcontexts of the main classification context. Before entering into the details, we give some new definitions.

Definition 8. *Let t be a set of values such that* $(\text{obj}(t), t)$ *is a test for G_+. The value $m \in M, m \in t$ is essential in t if* $(\text{obj}(t \setminus m), (t \setminus m))$ *is not a test for a given set of objects.*

Generally, we are interested in finding the maximal subset sbmax$(t) \subset t$ such that $(\text{obj}(t), t)$ is a test but $(\text{obj}(\text{sbmax}(t)), \text{sbmax}(t))$ is not a test for a given set of positive objects. Then sbmin$(t) = t \setminus$sbmax(t) is a minimal set of essential values in t.

Definition 9. *Let $s \subseteq G_+$, assume also that* $(s, \text{val}(s))$ *is not a test. The object g, $g \in s$ is said to be an essential in s if* $(s \setminus g, \text{val}(s \setminus g))$ *proves to be a test for a given set of positive objects.*

Generally, we are interested in finding the maximal subset $\mathrm{sbmax}(s) \subset s$ such that $(s, \mathrm{val}(s))$ is not a test but $(\mathrm{sbmax}(s), \mathrm{val}(\mathrm{sbmax}(s)))$ is a test for a given set of positive objects. Then $\mathrm{sbmin}(s) = s \setminus \mathrm{sbmax}(s)$ is a minimal set of essential objects in s.

Finding quasi-maximal (minimal) subsets of objects and values is the key procedure behind searching for initial content of STGOOD and determining the number of subtasks to be solved.

5.1 Using the Diagnostic Rule of the Second Type

An Approach for Searching for Initial Content of STGOOD. In the beginning of inferring GMRTs, the set STGOOD is empty. Next we describe the procedure to obtain an initial content of it. This procedure extracts a quasi-maximal subset $s_* \subseteq G_+$, which is the extent of a test for G_+ (maybe not good).

We begin with the first object g_1 of s_*, then we take the next object g_2 of s_* and evaluate the function $\mathrm{to\_be\_test}(\{g_1, g_2\}, \mathrm{val}(\{g_1, g_2\}))$. If the value of the function is $true$, then we take the next object g_3 of s_* and evaluate the function $\mathrm{to\_be\_test}(\{g_1, g_2, g_3\}, \mathrm{val}(\{g_1, g_2, g_3\}))$. If the value of the function $\mathrm{to\_be\_test}(\{g_1, g_2\}, \mathrm{val}(\{g_1, g_2\}))$ is $false$, then the object g_2 of s_* is skipped and the function $\mathrm{to\_be\_test}(\{g_1, g_3\}, \mathrm{val}(\{g_1, g_3\})))$ is evaluated. We continue this process until we achieve the last object of s_*. This procedure can be considered as a realization of the diagnostic rule of the second type.

To illustrate this procedure, we use the sets D_+ and D_- represented in Tab.2 and 3 (our illustrative example). In these tables $M = \{m_1, \ldots, m_{26}\}$. The set SPLUS_0 for positive class of objects is in Tab.4. The initial content of STGOOD_0 is {(2,10), (3, 10), (3, 8), (4, 12), (1, 4, 7), (1, 5,12), (2, 7, 8), (3, 7, 12), (1, 2, 12, 14), (2, 3, 4, 7), (4, 6, 8, 11)}.

In what follows, we denote subsets of values $\{m_8, m_9\}$, $\{m_{14}, m_{15}\}$ by m_a and m_b, respectively. Applying operation $\mathrm{generalization\_of}(s) = s'' = \mathrm{obj}(\mathrm{val}(s))$ to $\forall s \in$ STGOOD, we obtain STGOOD_1 = {(2,10), (3, 10), (3, 8), (4, 7, 12), (1, 4, 7), (1, 5,12), (2, 7, 8), (3, 7, 12), (1, 2, 12, 14), (2, 3, 4, 7), (4, 6, 8, 11)}.

By Th.1, we can delete value m_{12} from consideration, see $\mathrm{splus}(m_{12})$ in Tab.4. The initial content of STGOOD allows to decrease the number of using the procedure to_be_test() and the number of putting extents of tests into STGOOD.

The number of subtasks to be solved. This number is determined by the number of essential values in the set M. The quasi-minimal subset of essential values in M can be found by a procedure analogous to the procedure applicable to search for the initial content of STGOOD.

As a result of this procedure, we have quasi-maximal subset $\mathrm{sbmax}(M)$ of M, such that $(\mathrm{obj}(\mathrm{sbmax}(M)), \mathrm{sbmax}(M))$ is not a test for positive examples. Then subset $\mathrm{LEV} = M \setminus \mathrm{sbmax}(M)$ is quasi-minimal subset of essential values in M. We have the following LEV: $\{m_{16}, m_{18}, m_{19}, m_{20}, m_{21}, m_{22}, m_{23}, m_{24}, m_{26}\}$. Note, that searching for the number of subtasks to be solved is also based on using the diagnostic rule of the second type.

Table 2. The set D_+ of positive object descriptions

G	D_+
1	m_1 m_2 m_5 m_6 m_{21} m_{23} m_{24} m_{26}
2	m_4 m_7 m_8 m_9 m_{12} m_{14} m_{15} m_{22} m_{23} m_{24} m_{26}
3	m_3 m_4 m_7 m_{12} m_{13} m_{14} m_{15} m_{18} m_{19} m_{24} m_{26}
4	m_1 m_4 m_5 m_6 m_7 m_{12} m_{14} m_{15} m_{16} m_{20} m_{21} m_{24} m_{26}
5	m_2 m_6 m_{23} m_{24}
6	m_7 m_{20} m_{21} m_{26}
7	m_3 m_4 m_5 m_6 m_{12} m_{14} m_{15} m_{20} m_{22} m_{24} m_{26}
8	m_3 m_6 m_7 m_8 m_9 m_{13} m_{14} m_{15} m_{19} m_{20} m_{21} m_{22}
9	m_{16} m_{18} m_{19} m_{20} m_{21} m_{22} m_{26}
10	m_2 m_3 m_4 m_5 m_6 m_8 m_9 m_{13} m_{18} m_{20} m_{21} m_{26}
11	m_1 m_2 m_3 m_7 m_{19} m_{20} m_{21} m_{22} m_{26}
12	m_2 m_3 m_{16} m_{20} m_{21} m_{23} m_{24} m_{26}
13	m_1 m_4 m_{18} m_{19} m_{23} m_{26}
14	m_{23} m_{24} m_{26}

Table 3. The set D_- of negative object descriptions

G	D_-	G	D_-
15	$m_3m_8m_{16}m_{23}m_{24}$	32	$m_1m_2m_3m_7m_9m_{13}m_{18}$
16	$m_7m_8m_9m_{16}m_{18}$	33	$m_1m_5m_6m_8m_9m_{19}m_{20}m_{22}$
17	$m_1m_{21}m_{22}m_{24}m_{26}$	34	$m_2m_8m_9m_{18}m_{20}m_{21}m_{22}m_{23}m_{26}$
18	$m_1m_7m_8m_9m_{13}m_{16}$	35	$m_1m_2m_4m_5m_6m_7m_9m_{13}m_{16}$
19	$m_2m_6m_7m_9m_{21}m_{23}$	36	$m_1m_2m_6m_7m_8m_{13}m_{16}m_{18}$
20	$m_{19}m_{20}m_{21}m_{22}m_{24}$	37	$m_1m_2m_3m_4m_5m_6m_7m_{12}m_{14}m_{15}m_{16}$
21	$m_1m_{20}m_{21}m_{22}m_{23}m_{24}$	38	$m_1m_2m_3m_4m_5m_6m_9m_{12}m_{13}m_{16}$
22	$m_1m_3m_6m_7m_9m_{16}$	39	$m_1m_2m_3m_4m_5m_6m_{14}m_{15}m_{19}m_{20}m_{23}m_{26}$
23	$m_2m_6m_8m_9m_{14}m_{15}m_{16}$	40	$m_2m_3m_4m_5m_6m_7m_{12}m_{13}m_{14}m_{15}m_{16}$
24	$m_1m_4m_5m_6m_7m_8m_{16}$	41	$m_2m_3m_4m_5m_6m_7m_9m_{12}m_{13}m_{14}m_{15}m_{19}$
25	$m_7m_{13}m_{19}m_{20}m_{22}m_{26}$	42	$m_1m_2m_3m_4m_5m_6m_{12}m_{16}m_{18}m_{19}m_{20}m_{21}m_{26}$
26	$m_1m_2m_3m_5m_6m_7m_{16}$	43	$m_4m_5m_6m_7m_8m_9m_{12}m_{13}m_{14}m_{15}m_{16}$
27	$m_1m_2m_3m_5m_6m_{13}m_{18}$	44	$m_3m_4m_5m_6m_8m_9m_{12}m_{13}m_{14}m_{15}m_{18}m_{19}$
28	$m_1m_3m_7m_{13}m_{19}m_{21}$	45	$m_1m_2m_3m_4m_5m_6m_7m_8m_9m_{12}m_{13}m_{14}m_{15}$
29	$m_1m_4m_5m_6m_7m_8m_{13}m_{16}$	46	$m_1m_3m_4m_5m_6m_7m_{12}m_{13}m_{14}m_{15}m_{16}m_{23}m_{24}$
30	$m_1m_2m_3m_6m_{12}m_{14}m_{15}m_{16}$	47	$m_1m_2m_3m_4m_5m_6m_8m_9m_{12}m_{14}m_{16}m_{18}m_{22}$
31	$m_1m_2m_5m_6m_{14}m_{15}m_{16}m_{26}$	48	$m_2m_8m_9m_{12}m_{14}m_{15}m_{16}$

Table 4. The set SPLUS_0

$\text{splus}(m), m \in M$	$\text{splus}(m), m \in M$
$\text{splus}(m_a) \to \{2, 8, 10\}$	$\text{splus}(m_{22}) \to \{2, 7, 8, 9, 11\}$
$\text{splus}(m_{13}) \to \{3, 8, 10\}$	$\text{splus}(m_{23}) \to \{1, 2, 5, 12, 13, 14\}$
$\text{splus}(m_{16}) \to \{4, 9, 12\}$	$\text{splus}(m_3) \to \{3, 7, 8, 10, 11, 12\}$
$\text{splus}(m_1) \to \{1, 4, 11, 13\}$	$\text{splus}(m_4) \to \{2, 3, 4, 7, 10, 13\}$
$\text{splus}(m_5) \to \{1, 4, 7, 10\}$	$\text{splus}(m_6) \to \{1, 4, 5, 7, 8, 10\}$
$\text{splus}(m_{12}) \to \{2, 3, 4, 7\}$	$\text{splus}(m_7) \to \{2, 3, 4, 6, 8, 11\}$
$\text{splus}(m_{18}) \to \{3, 9, 10, 13\}$	$\text{splus}(m_{24}) \to \{1, 2, 3, 4, 5, 7, 12, 14\}$
$\text{splus}(m_2) \to \{1, 5, 10, 11, 12\}$	$\text{splus}(m_{20}) \to \{4, 6, 7, 8, 9, 10, 11, 12\}$
$\text{splus}(m_b) \to \{2, 3, 4, 7, 8\}$	$\text{splus}(m_{21}) \to \{1, 4, 6, 8, 9, 10, 11, 12\}$
$\text{splus}(m_{19}) \to \{3, 8, 9, 11, 13\}$	$\text{splus}(m_{26}) \to \{1, 2, 3, 4, 6, 7, 9, 10, 11, 12, 13, 14\}$

Proposition 1. *Each essential value is included at least in one positive object description.*

Proof. Assume that for an object description $t_g, g \in G_+$, we have $t_g \cap LEV = \emptyset$. Then $t_g \subseteq M \setminus \text{LEV}$. But $M \setminus \text{LEV}$ is included at least in one of the negative object descriptions and, consequently, t_g also possesses this property. But it contradicts to the fact that t_g is the description of a positive object. □

Proposition 2. *Assume that $X \subseteq M$. If $X \cap \text{LEV} = \emptyset$, then* $\text{to\_be\_test}(\text{obj}(X), X) = \mathit{false}$.

Proposition 3. *Let $\mathbb{K}_\pm$ be a given classification context. For finding all GRMTs containing in $\mathbb{K}_\pm$, it is sufficient to solve this problem only for subcontexts associated with essential values in M (the set LEV).*

Proposition 2 is the consequence of Proposition 1. The poof of Proposition 3 is obvious.

Note that the description of $t_{14} = \{m_{23}, m_{24}, m_{26}\}$ is closed because of $\text{obj}\{m_{23}, m_{24}, m_{26}\} = \{1, 2, 12, 14\}$ and $\text{val}\{1, 2, 12, 14\} = \{m_{23}, m_{24}, m_{26}\}$. We also know that $s = \{1, 2, 12, 14\}$ is closed too (we obtained this result during generalization of elements of STGOOD). So $(\text{obj}(\{m_{23}, m_{24}, m_{26}\}), \{m_{23}, m_{24}, m_{26}\})$ is a maximally redundant test for positive objects and we can, consequently, delete t_{14} from consideration. As a result of deleting m_{12} and t_{14}, we have the modified set SPLUS (Tab.5).

5.2 Using the Rule of Alternative for Splitting Subcontexts

Essentially, searching for GMRTs turns out a process of generating deductive reasoning rules of the first type, which are used immediately during this process. Now we shall give an example of constructing and using the rules of alternative (forbidden rules).

Table 5. The set SPLUS$_1$

splus$(m), m \in M$	splus$(m), m \in M$
splus$(m_a) \rightarrow \{2, 8, 10\}$	splus$(m_{22}) \rightarrow \{2, 7, 8, 9, 11\}$
splus$(m_{13}) \rightarrow \{3, 8, 10\}$	splus$(m_{23}) \rightarrow \{1, 2, 5, 12, 13\}$
splus$(m_{16}) \rightarrow \{4, 9, 12\}$	splus$(m_3) \rightarrow \{3, 7, 8, 10, 11, 12\}$
splus$(m_1) \rightarrow \{1, 4, 11, 13\}$	splus$(m_4) \rightarrow \{2, 3, 4, 7, 10, 13\}$
splus$(m_5) \rightarrow \{1, 4, 7, 10\}$	splus$(m_6) \rightarrow \{1, 4, 5, 7, 8, 10\}$
	splus$(m_7) \rightarrow \{2, 3, 4, 6, 8, 11\}$
splus$(m_{18}) \rightarrow \{3, 9, 10, 13\}$	splus$(m_{24}) \rightarrow \{1, 2, 3, 4, 5, 7, 12\}$
splus$(m_2) \rightarrow \{1, 5, 10, 11, 12\}$	splus$(m_{20}) \rightarrow \{4, 6, 7, 8, 9, 10, 11, 12\}$
splus$(m_b) \rightarrow \{2, 3, 4, 7, 8\}$	splus$(m_{21}) \rightarrow \{1, 4, 6, 8, 9, 10, 11, 12\}$
splus$(m_{19}) \rightarrow \{3, 8, 9, 11, 13\}$	splus$(m_{26}) \rightarrow \{1, 2, 3, 4, 6, 7, 9, 10, 11, 12, 13\}$

Form all the pairs of objects $(i, j) \subseteq G_+$; form a set FR of forbidden pairs of objects, FR $= \{(i, j) :$ to_be_test$((i, j), \text{val}(i, j))$ is $false\}$ (Tab. 6); then the set of admissible pairs is ADM $= \{(i, j) :$ to_be_test$((i, j), \text{val}(i, j))$ is $true\}$ (Tab. 6).

Each forbidden pair (i, j) generates rule of alternative: j OR $i \rightarrow true(always)$; $i, j \rightarrow false$. This rule says that i and j cannot simultaneously enter in any extent of GMRTs, either i or j but not both.

These rules of alternatives allow splitting any splus$(m), m \in M$ into subsets not containing any forbidden pair of objects. An example is in Tab. 7.

As a result of splitting, we shall obtain the splitting tree, leaves of which are subsets of objects not containing any prohibited object pairs. Subcontexts for GMRTs inferring are leaves of the tree with the number of objects more or equal to 4. So, in Tab. 7, we have only one subcontext (4,7,8,11) for searching for tests. As far as triplets of objects, if a triplet is not the extent of a test, then we have all information about pairs of objects contained in it.

An analysis of set ADM leads to using some inductive reasoning rules [19]. If element j enters in s_{q+1}, then it must enter in q proper subsets of s_{q+1}, where s_{q+1} is a set containing $(q + 1)$ elements. If we observe that j enters in only one doublet (pair), then it cannot enter in any triplet. If we observe that j enters in only one triplet, then it cannot enter in any quadruplet and so on. If an element enters in two and only two doublets, it means that it will enter only in one triplet. If an element enters in three and only three doublets, it can enter in only one quadruplet.

This inductive reasoning is applicable to constructing triplets from doublets, quadruplets from triplets and so on. In our example: Object 9 enters in one and only one doublet, hence (9,11) cannot be included in any triplet. We conclude that (9,11) is the extent of a GMRT. Object 5 enters in two and only two pairs, hence it is included in only one triplet (1,5,12) already contained in the initial content of STGOOD. Object 5 cannot be included in any quadruplet. Object 6

Table 6. Classification of object pairs

g	Admissible pairs of objects	Forbidden pairs of objects
1	(1,2) (1,4) (1,5) (1,7) (1,12)	(1,3) (1,6) (1,8) (1,9) (1,10) (1,11) (1,13)
2	(2,1) (2,3) (2,4) (2,7) (2,8) (2,10) (2,12)	(2,5) (2,6) (2,9) (2,11) (2,13)
3	(3,2) (3,4) (3,7) (3,8) (3,10) (3,11) (3,12)	(3,1) (3,5) (3,6) (3,9) (3,13)
4	(4,1) (4,2) (4,3) (4, 6) (4,7) (4,8) (4,11) (4,12)	(4,5) (4,9) (4,10) (4,13)
5	(5,1) (5,12)	(5,2) (5,3) (5,4) (5,6) (5,7) (5,8) (5,9) (5,10) (5,11) (5,13)
6	(6,4) (6,8) (6,11)	(6,1) (6,2) (6,3) (6,5) (6,7) (6,9) (6,10) (6,12) (6,13)
7	(7,1) (7,2) (7,3) (7,4) (7,8) (7,11) (7,12)	(7,5) (7, 6) (7,9) (7,10) (7,13)
8	(8,2)(8,3)(8,4) (8,6)(8,7) (8,10) (8,11)	(8,1) (8,5) (8,9) (8,12) (8,13)
9	(9,11)	All the other pairs
10	(10,2) (10,3) (10,8)	(10,1) (10,4) (5,10) (10,6) (10,7) (10,9) (10,11) (10,12) (10,13)
11	(11,3) (11,4) (6,11) (7,11) (11,9)	(11,1) (11,2) (5,11) (11,9) (11, 10) (11, 12) (11,13)
12	(12,1) (12,2) (12,3) (12,4) (12,5) (12,7)	(12,6) (12,8) (12,9) (12,10) (12,11) (12,13)
13	∅	All the pairs

Table 7. An example of splitting splus(m_2)

No	splus(m_2)	Forbidden pair
1	(4,7,8,11,12)	(8,12)
2	(4,7,8,11), (4,7,11,12)	
3	(4,7,11,12)	(11,12)
4	(4,7,11) (4,7,12)	

enters in three and only three pairs, hence it is included in only one quadruplet (4,6,8,11), already contained in STGOOD. Object 13 is not included in any admissible pair of objects. We put $\{9, 11\}$ and $\{13\}$ into STGOOD and delete objects 9, 5, 6, and 13 from consideration. This action implies deleting values m_{16}, m_{18}, and m_{23} (see, please Tab. 8), because splus(m_i), where i take values 16, 18 or 23, are absorbed by some elements of STGOOD.

Table 8. The set SPLUS$_2$

splus(m), $m \in M$	splus(m), $m \in M$
splus(m_a) $\to \{2, 8, 10\}$	splus(m_{22}) $\to \{2, 7, 8, 11\}$
splus(m_{13}) $\to \{3, 8, 10\}$	**splus(m_{23}) $\to \{1, 2, 12\}$**
splus(m_{16}) $\to \{4, 12\}$	splus(m_3) $\to \{3, 7, 8, 10, 11, 12\}$
splus(m_1) $\to \{1, 4, 11\}$	splus(m_4) $\to \{2, 3, 4, 7, 10, 13\}$
splus(m_5) $\to \{1, 4, 7, 10\}$	splus(m_6) $\to \{1, 4, 7, 8, 10\}$
	splus(m_7) $\to \{2, 3, 4, 6, 8, 11\}$
splus(m_{18}) $\to \{3, 10\}$	splus(m_{24}) $\to \{1, 2, 3, 4, 7, 12\}$
splus(m_2) $\to \{1, 10, 11, 12\}$	splus(m_{20}) $\to \{4, 7, 8, 10, 11, 12\}$
splus(m_b) $\to \{2, 3, 4, 7, 8\}$	splus(m_{21}) $\to \{1, 4, 8, 9, 10, 11, 12\}$
splus(m_{19}) $\to \{3, 8, 11\}$	splus(m_{26}) $\to \{1, 2, 3, 4, 7, 10, 11, 12\}$

We analyze triplets in SPLUS: splus(m_*) $= \{2, 8, 10\}$, splus(m_1) $= \{1, 4, 11\}$, splus(m_{19}) $= \{3, 8, 11\}$, and splus(m_{13}) $= \{3, 8, 10\}$. As a result, we obtain the extents of two new GMRTs $\{3, 11\}$ and $\{8, 10\}$. After that, we can delete object 10. Then we can delete values m_5, m_2, and m_4 after analyzing their modified splus. Now we split splus(m_{22}), splus(m_{20}), splus(m_{21}), splus(m_{24}), and splus(m_{26}), because these values are the only remaining essential values in the set LEV. Tab. 9 shows the number of subtasks in splitting trees for these essential values and new obtained tests.

Table 9. Splitting splus(m), m is essential value

splus(m)	The number of subtasks	New tests	Deleted elements
splus(m_{22})		$\{7, 8, 11\}$	m_{26}
splus(m_{20})	1		m_{20}
splus(m_{21})			m_{21}, t_8
splus(m_{24})	2		m_{24}, t_1, t_{12}, t_7; remaining set of indices $\{2, 3, 4\}$ is absorbed by element $\{2, 3, 4, 7\}$ in STGOOD

Some words about future investigations. The most important mathematical tasks worth solving are the following:

1. For irredundant splus(m) to determine whether it contains extents of new GMRTs;
2. Optimization of selecting forbidden pairs of objects in order not to obtain repetition of subsets in splitting trees.

6 Conclusion

A sketch of classification reasoning is given in the paper. The key notions of the reasoning are notions of classification and its good approximations based on good diagnostic tests. The good diagnostic tests have the dual nature: they generate functional and implicative dependencies and, in the same time, sets of objects satisfying these dependencies. The dependencies are logical assertions on the base of which deductive phase of classification reasoning is realized.

In the paper, we deal only with GMRTs as value implications. An algorithm for inferring GMRTs from a set of data is given. A decomposition of the main task of inferring GMRTs into subtasks associated with the certain types of subcontexts is introduced.

There are various ways of using the proposed decompositions and extracted initial information about GMRTs, but we confine ourselves to the most important ideas allowing to obtain not only tests (and corresponding knowledge) but also, simultaneously, contexts inside of which these tests have been inferred. Currently, there is considerable interest in contextual information search and representation for using it to improve web search results or query expansion [3,4,10]. Classification reasoning can serve as a basic tool for constructing and using content-dependent knowledge in different problem domains and applications.

References

1. Baixeries, J., Kaytoue, M., Napoli, A.: Computing functional dependencies with pattern structures. In: Szathmary, L., Priss, U. (eds.) Proceedings of The Ninth International Conference on Concept Lattices and Their Applications. CEUR Workshop Proceedings, vol. 972, pp. 175–186. CEUR-WS.org (2012)
2. Banerji, R.B.: Theory of problem solving: an approach to artificial intelligence, vol. 17. Elsevier Publishing Company (1969)
3. Bankar, S., Nagpure, R.: Contextual information search based on domain using query expansion. In: International Journal of Emerging Trends and Technology in Computer Science. vol. 3, pp. 148 – 151 (2014)
4. Bouramoul, A., Kholladi, M.K., Doan, B.L.: PRESY: A Context Based Query Reformulation Tool for Information Retrieval on the Web. Journal of Computer Science 6(4), 470–477 (2010)
5. Ferré, S., Ridoux, O.: The use of associative concepts in the incremental building of a logical context. In: Priss, U., Corbett, D., Angelova, G. (eds.) ICCS. Lecture Notes in Computer Science, vol. 2393, pp. 299–313. Springer (2002)

6. Finn, V.: The synthesis of cognitive procedures and the problem of induction. NTI ser.2, 1-2, 8–44 (1999), (in Russian)
7. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer, Berlin (1999)
8. Ganter, B., Kuznetsov, S.O.: Formalizing hypotheses with concepts. In: Proceedings of the Linguistic on Conceptual Structures: Logical Linguistic, and Computational Issues. pp. 342–356. Springer-Verlag (2000)
9. Ganter, B., Kuznetsov, S.: Pattern structures and their projections. In: Delugach, H., Stumme, G. (eds.) Conceptual Structures: Broadening the Base, Lecture Notes in Computer Science, vol. 2120, pp. 129–142. Springer Berlin Heidelberg (2001)
10. Klimešová, D., Ocelíková, E.: Study on context understanding, knowledge transformation and decision support systems. WSEAS Trans. Info. Sci. and App. 7(7), 985–994 (Jul 2010)
11. Kuznetsov, S.: Machine learning on the basis of formal concept analysis. Automation and Remote Control 62(10), 1543–1564 (2001)
12. Mill, J.S.: The System of Logic Ratiocinative and Inductive Being a Connected View of the Principles of Evidence, and the Methods of Scientific Investigation. Longmans (1872)
13. Naidenova, K.: Reducing one class of machine learning algorithms to logical operations of plausible reasoning. Journal of Computer and Systems Sciences International 48(3), 401–414 (2009)
14. Naidenova, X.A., Plaksin, M.V., Shagalov, V.L.: Inductive Inferring All Good Classification Tests. In: Valkman, J. (ed.) "Knowledge-Dialog-Solution", Proceedings of International Conference. vol. 1, pp. 79 – 84. Kiev Institute of Applied Informatics, Kiev, Ukraine (1995)
15. Naidenova, X.: Machine Learning as a diagnostic task. In: Arefiev, I. (ed.) Knowledge-Dialog-Solution, Materials of the Short-Term Scientific Seminar, pp. 26 – 36. State North-West Technical University Press, St Petersburg, Russia (1992)
16. Naidenova, X.: The Data-Knowlege Transformation. In: Solovyev, V. (ed.) Text Processing and Cognitive Technologies, vol. 3, pp. 130–151. Pushchino (1999)
17. Naidenova, X.: Good classification tests as formal concepts. In: Domenach, F., Ignatov, D., Poelmans, J. (eds.) Formal Concept Analysis, Lecture Notes in Computer Science, vol. 7278, pp. 211–226. Springer Berlin Heidelberg, Leuven (2012)
18. Naidenova, X., Ermakov, A.: The decomposition of good diagnostic test inferring algorithms. In: Alty, J., Mikulich, L., Zakrevskij, A. (eds.) "Computer-Aided Design of Discrete Devices" (CAD DD2001), Proceedings of the 4-th Inter. Conf., vol. 3, pp. 61 – 68. Minsk (2001)
19. Naidenova, X.: An incremental learning algorithm for inferring logical rules from examples in the framework of the common reasoning process. In: Triantaphyllou, E., Felici, G. (eds.) Data Mining and Knowledge Discovery Approaches Based on Rule Induction Techniques, Massive Comp., vol. 6, pp. 89–147. Springer (2006)
20. Naidenova, X.A., Parkhomenko, V.: Attributive and object subcontexts in inferring good maximally redundant tests. In: Bertet, K., Rudolph, S. (eds.) Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014. CEUR Workshop Proceedings, vol. 1252, pp. 181–193. CEUR-WS.org (2014)
21. Ore, O.: Galois connections. Trans. Amer. Math. Soc 55, 494–513 (1944)
22. Rasiowa, H.: An Algebraic Approach to Non-classical Logics. Studies in logic and the foundations of mathematics, North-Holland Publishing Company (1974)

Concept Lattices of RDF Graphs

Jens Kötters

Abstract. The concept lattice of an RDF graph is defined. The intents are described by graph patterns rather than sets of attributes, a view that is supported by the fact that RDF data is essentially a graph. A simple formalization by triple graphs defines pattern closures as connected components of graph products. The patterns correspond to conjunctive queries, generalization of properties is supported. The relevance of the notion of connectedness is discussed, showing the impact of ontological considerations on the process of concept formation. Finally, definitions are given which allow the construction of pattern closures of concept graphs, which formalize Conceptual Graphs.

Keywords: RDF Schema, Pattern Concepts, Conceptual Graphs, Navigation

1 Introduction

The Resource Description Framework (RDF) is an extensible standard for knowledge representation which relies on the use of simple sentences, each consisting of a subject, a predicate and an object. In RDF terminology, such a sentence is called a *triple*; a collection of triples is called an *RDF graph*, and the entities which occur as subjects, predicates or objects of triples (i.e.,the things being talked about) are called *resources*. Figure 1 shows an RDF graph in the text-based Turtle notation [5] (namespace declarations are omitted). Figure 2 shows the same RDF graph, using a standard graphical notation (see e.g. [9]). Each triple is represented by an arc. In the abundance of such data, query languages, most notably SPARQL [1], can be used to gain access to the information contained. Querying alone is however of limited use to anyone who needs information but does not know specifically what to look for, or how to ask for it, or maybe whether such information can be found at all. In such cases, concept lattices can in principle guide the exploration of the data source, as they support interactive and data-aware query modification. In connection with SPARQL, this has already been demonstrated in [7]. The current paper is also motivated by data exploration, but deliberately limits the query language to conjunctive queries. In this case, the entire search space is obtained as a concept lattice in which each intent is described by a single *most specific query* (MSQ).

Conjunctive queries can be represented as graphs, and entailment is described by graph homomorphisms [6]. Figure 3 shows a graph representing a conjunctive query – "Someone who spoke about a sailor (in the subject)" – and its SPARQL translation below, which can always be obtained in a straightforward way. Graph queries are modeled after RDF graphs (Fig.2), so that solutions are

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors. Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

```
ex:Frank   ex:loves     ex:Mary .
ex:Frank   rdf:type     ex:Sailor .
ex:Frank   rdf:type     ex:Englishman .
ex:Tom     ex:loves     ex:Mary .
ex:Tom     rdf:type     ex:Gentleman .
ex:Tom     ex:dislikes  ex:Frank .
ex:Frank   ex:dislikes  ex:Tom .
ex:Tom     ex:name      "Tom" .
ex:Tom     ex:said      ex:S1 .
ex:S1      rdf:subj     ex:Tom .
ex:S1      rdf:pred     ex:name .
ex:S1      rdf:obj      "Tom" .
ex:Frank   ex:jested    ex:S2.
ex:S2      rdf:subj     ex:Frank .
ex:S2      rdf:pred     ex:likes .
ex:S2      rdf:obj      ex:Lobscouse .

ex:Liam    ex:loves     ex:Aideen .
ex:Aideen  ex:loves     ex:Liam .
ex:Rowan   ex:loves     ex:Aideen .
ex:Rowan   ex:hates     ex:Liam .
ex:Aideen  ex:said      ex:S3 .
ex:S3      rdf:subj     ex:Rowan .
ex:S3      rdf:pred     ex:likes .
ex:S3      rdf:obj      ex:Aideen .
```

Fig. 1. Sample RDF graph in Turtle notation

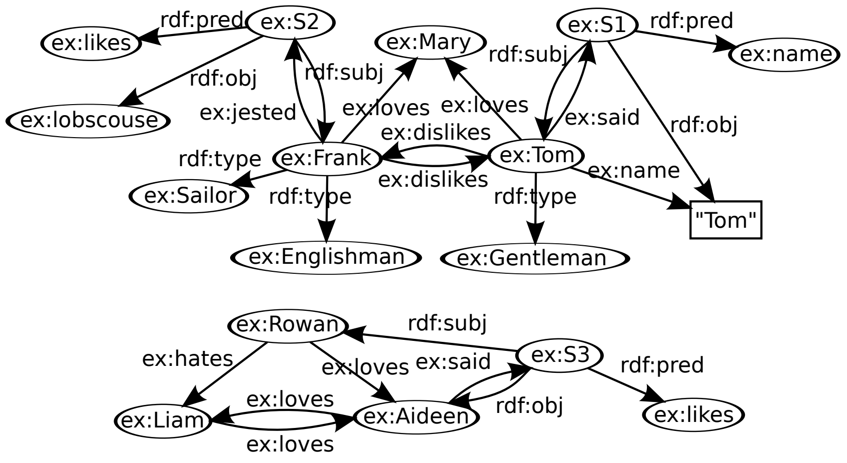
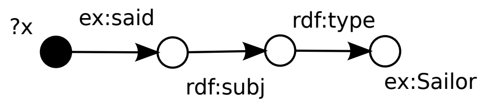


Fig. 2. Drawing of the RDF graph of Fig.1

described by homomorphisms, as well. But there are some differences to RDF graphs. Firstly, all nodes (and all arcs) represent variables. The subject(s) of the query are distinguished in some way (in Fig. 3, a black node represents the single subject), whereas all other variables are implicitly understood as being existentially quantified. Finally, the labels are attributes of a formal context $\mathbb{K}$ (for the example, see Fig. 4), and thus represent formal concepts of its concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$. In practice, the formal context may encode background knowledge from an RDF Schema file like the one in Fig. 5. The query vocabulary is restricted to the attributes of $\mathbb{K}$, the name `Frank` e.g. may not be used in a query for the given example.



```
SELECT ?x WHERE { ?x ex:said ?y. ?y rdf:subj ?z. ?z rdf:type ex:Sailor. }
```

Fig. 3. Conjunctive query as a graph (above) and in SPARQL (below)

For the rest of the paper, we shall refer to queries as patterns. Sect. 2 introduces triple graphs as formalizations of queries without distinguished variables (they correspond to SPARQL `WHERE`-clauses), and a query in one distinguished variable is then represented by the pair (x, G) consisting of a triple graph G and a vertex x of G. The case of queries in several variables is not treated in this paper, but has been treated in [11] in connection with relational structures. Section 3 shows how the search space for conjunctive queries over an RDF graph is obtained as a concept lattice, where concept intents are patterns rather than sets of attributes.

Lattices of *pattern concepts* were introduced in [8], and although the theory presented there does not make specific assumptions of what a pattern is, it seems clear that graphs were meant to play a prominent role. In fact, the paper already presents an example application where patterns are chemical compounds and compares them by labeled graph homomorphisms. While chemical compounds are described by connected graphs, their supremum in terms of these homomorphisms is not necessarily connected (*ibid.*, pp. 139-140). Another suggestion made in [8] is to use Conceptual Graphs (CGs) as patterns. If graph patterns are used to describe situations (where several objects combine to a whole by means of the way they are related to each other), we would probably expect them to be connected. But if we formalize CGs using the same kind of labeled graphs that was used for chemical compounds, the set of patterns is generally not closed under suprema because, as we have seen, the supremum operation does not preserve connectedness.

Properties and Classes	name	type	loves	likes	hates	dislikes	jested	said	subj	pred	obj	Englishman	Gentleman	Man	Sailor	Person	”Tom”	lobscouse	Resource
name	×																		×
type		×																	×
loves			×	×															×
likes				×															×
hates					×	×													×
dislikes						×													×
jested							×	×											×
said								×											×
subj									×										×
pred										×									×
obj											×								×
Englishman												×		×		×			×
Gentleman													×	×		×			×
Man														×		×			×
Sailor															×	×			×
Person																×			×
”Tom”																	×		×
lobscouse																		×	×
Resource																			×

Fig. 4. Formal context $\mathbb{K}$ of properties and classes

```
ex:Englishman  rdfs:subClassOf     ex:Man
ex:Gentleman   rdfs:subClassOf     ex:Man
ex:Sailor      rdfs:subClassOf     ex:Person
ex:Man         rdfs:subClassOf     ex:Person
ex:loves       rdfs:subPropertyOf  ex:likes
ex:hates       rdfs:subPropertyOf  ex:dislikes
ex:jested      rdfs:subPropertyOf  ex:said
ex:loves       rdfs:domain         ex:Person
ex:loves       rdfs:range          ex:Person
```

Fig. 5. An RDFS ontology

The situation changes if graphs have a distinguished element. The supremum is a graph product, again with a distinguished element, and although the product of connected graphs is generally not connected, it makes sense to identify the supremum with the connected component which holds the distinguished variable. The relevance of connectedness is discussed in Sect. 4. In Sect. 5, the approach is applied to concept graphs.

2 Triple Graphs

A *triple graph* over $\mathbb{K}$ is a triple (V, T, κ), where V is a set of vertices, $T \subseteq V \times V \times V$ and $\kappa : V \to \underline{\mathfrak{B}}(\mathbb{K})$.

A morphism $\varphi : (V_1, T_1, \kappa_1) \to (V_2, T_2, \kappa_2)$ of triple graphs is a map $\varphi : V_1 \to V_2$ with

$$(x, y, z) \in T_1 \Rightarrow (\varphi x, \varphi y, \varphi z) \in T_2, \tag{1}$$

$$\kappa_1(x) \geq \kappa_2(\varphi x) \tag{2}$$

for all $x, y, z \in V_1$.

The product of a family of triple graphs $G_i = (V_i, T_i, \kappa_i)$, $i \in I$, is the triple graph

$$\prod_{i\in I} G_i := (\bigtimes_{i\in I} V_i,\ \{\ t^{\mathrm{T}} \mid t \in \bigtimes_{i\in I} T_i\},\ \bigvee_{i\in I} \kappa_i), \tag{3}$$

where $t^{\mathrm{T}} := ((t_{i0})_{i\in I}, (t_{i1})_{i\in I}, (t_{i2})_{i\in I})$ and $(\bigvee_{i\in I} \kappa_i)(v) := \bigvee_{i\in I} \kappa_i(v_i)$.

Finally, an RDF graph with the set T of triples is represented by the triple graph (R, T, γ^*), where R is the set of resources (properties and classes are duplicated so that they appear in only one triple) and

$$\gamma^*(g) := \begin{cases} (g'', g') \text{ if } g \in G, \\ \top := (G, G') \text{ otherwise} \end{cases} . \tag{4}$$

A *pattern* is a pair (v, H), where $H =: (V, T, \kappa)$ is a triple graph and $v \in V$. A *pattern morphism* $\varphi : (v_1, H_1) \to (v_2, H_2)$ is a morphism $\varphi : H_1 \to H_2$ with $\varphi(v_1) = v_2$. A *solution* is a pattern in the data graph. Pattern morphisms can thus describe solutions as well as entailment.

The product of a family of patterns is given by

$$\prod_{i\in I}(v_i, H_i) := ((v_i)_{i\in I}, \prod_{i\in I} H_i). \tag{5}$$

The definitions above follow a category theoretical approach. Queries (minus distinguished variable(s)) and the data source have representations in the same category (using $\varDelta$ for the data source), and morphisms $\lambda : G \to \varDelta$ can be understood as solutions of the respective query. Query entailment is then described by morphisms, and if the category has products, then so has the category of structured arrows (these are the pairs (ν, G), cf. [2]), which model queries with distinguished variables. The pattern closures (MSQs) arise as products from the set of possible solutions $(\lambda, \varDelta)$. For convenience, we write (x, G) instead of (ν, G) if the domain of ν is a one-element set (x being the image of that element).

3 Example RDF Graph and Concept Lattice

In this section, we will walk through the process of generating a concept lattice for the RDF graph in Fig. 1. To keep the example small (and perhaps meaningful), only person concepts will be created, i.e. concepts with extents in the set $\{\texttt{Tom}, \texttt{Frank}, \texttt{Mary}, \texttt{Rowan}, \texttt{Liam}, \texttt{Aideen}\}$.

As a first step, we determine the object intent for each person, i.e. its most complete description. For each person, its connected component in Fig. 2 doubles as a most complete description if we reinterpret the nodes and arcs as (distinct) variables, reinterpret the labels beside them as concept names (rather than resource names), and mark the variable corresponding to that person as distinguished. Let G_1 and G_2 denote, top to bottom, the components in Fig. 2 after reinterpretation, and let T, F, M, R, L and A denote the variables which replace `Tom`, `Frank`, `Mary`, `Rowan`, `Liam` and `Aideen`. Then the pairs (T, G_1), (F, G_1), (M, G_1), (R, G_2), (L, G_2) and (A, G_2) formalize the object intents. These pairs are considered patterns; each pattern consists of a distinguished variable and a triple graph (formalized in Sect. 2).

The person concepts are obtained from graph products of the six base patterns. Many concept intents have the same underlying triple graph (consider e.g. the base patterns), so there would be a lot of redundancy in the concept lattice if we were to draw intents next to each concept. A different kind of diagram avoids this redundancy while still showing all concepts and their intents : Fig. 6 shows all triple graphs which occur in the intents of person concepts, ordered by homomorphism. Choosing any of the nodes as a distinguished variable gives rise to a pattern intent, and this way concepts can be identified with triple graph nodes. The gray nodes in Fig. 6 are the person concepts (the white nodes are non-person concepts which occur in the descriptions of person concepts). The concept extents for the person concepts are drawn next to the gray nodes. The person concept lattice itself can be easily obtained from the diagram in Fig. 6 by connecting the gray nodes according to their extents.

Note however that some concepts have multiple representations in Fig. 6. These may occur in the same triple graph, which indicates symmetry (i.e., a triple graph automorphism). An example is the pair of lovers on the right side of Fig. 6. The other case is where the same concept occurs in different triple graphs. An example would be the Mary concept, which occurs in the lower left triple graph as well as in the triple graph above it on the left side of Fig. 6. In such cases, there is always a most specific triple graph containing that concept; it describes that concept's pattern intent. Other triple graphs containing the same concept can be folded onto the most specific triple graph (which is contained as a subgraph). But the fact that triple graphs may reappear as subgraphs of other triple graphs translates into another redundancy problem when drawing the hierarchy of triple graph product components. In Fig. 6, this kind of redundancy has been avoided by cutting those subgraphs out and indicate the glue nodes (for the gluing operation which inverts the cutting operation) with an asterisk. The glue nodes are only marked in the upper triple graph, not in the lower triple graph, because the lower triple graph can be considered self-contained (a

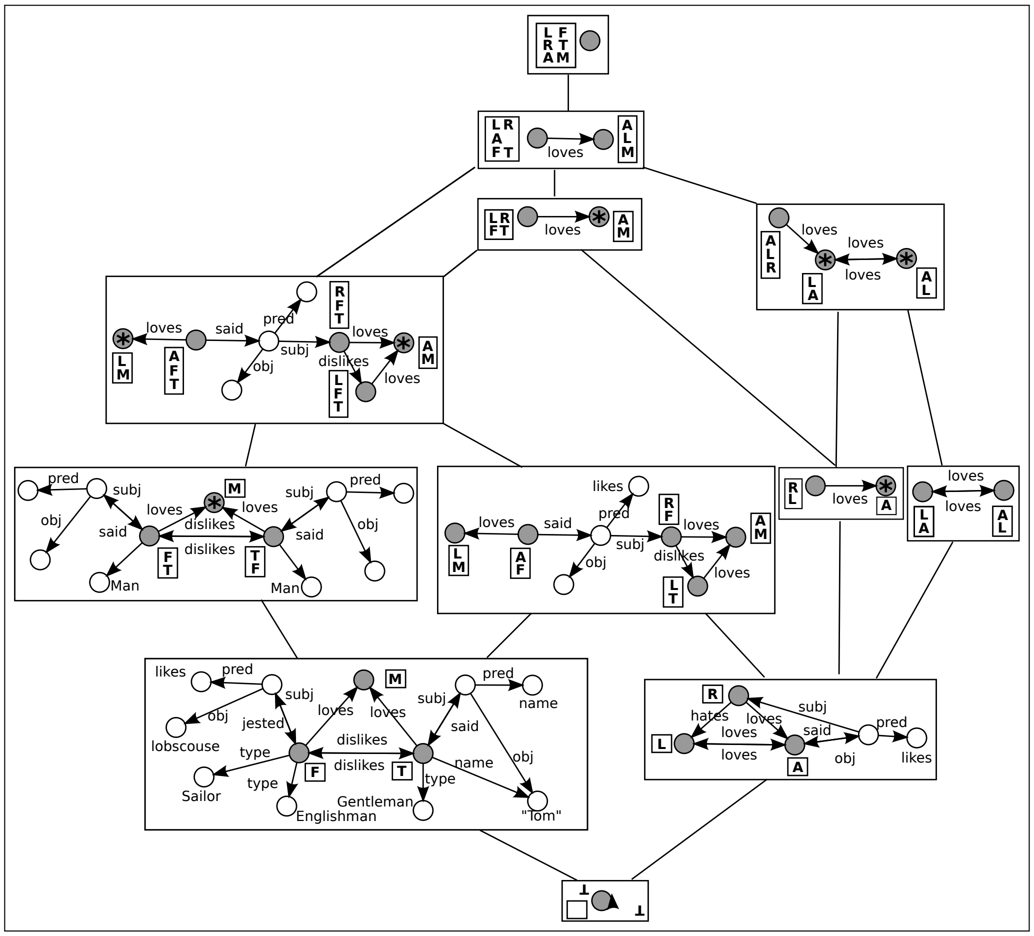


Fig. 6. Concepts of the RDF graph

glue node's counterpart in the lower graph can be identified by its extent). To put this differently: in every triple graph, the asterisk concepts are part of the description of the non-asterisk concepts, but not vice versa.

The extent of a concept reveals how its pattern intent can be obtained. Consider for example the concept with extension $\{R, F, T\}$, which is part of the top left triple graph. The extension tells us that the pattern intent is the (core graph of the) component of the pattern product $(R, G_2) \times (F, G_1) \times (T, G_1)$, which contains the vertex (R, F, T), and (R, F, T) becomes the designated variable. The diagram also shows that $\{R, T\}$ is a generating set for this extent, because it is not contained in a concept extent of the triple graph's lower neighbors, and so $(R, G_2) \times (T, G_1)$ already produces the same pattern. As was described in [8], Ganter's NextConcept algorithm can be used to systematically generate all extents (and thus all concepts), computing pattern products in the process. It is advisable to use graph folding to obtain minimal equivalent patterns, for output as well as inbetween computational steps.

Product patterns could also be computed directly from the triples in Turtle notation (Fig. 1), by combining triples with each other. Informally, we could write triple graphs as sets of triples $(x : \kappa(x), p : \kappa(p), y : \kappa(y))$ (borrowing from RDF and CG notation), and compute the product triples by taking suprema componentwise, like so:

$$\begin{array}{ll} & (F : \top, p : \text{type}, y_1 : \text{Englishman}) \\ \vee & (T : \top, p : \text{type}, y_2 : \text{Gentleman}) \\ = & ([\begin{smallmatrix} F \\ T \end{smallmatrix}] : \top, [\begin{smallmatrix} p \\ p \end{smallmatrix}] : \text{type}, [\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}] : \text{Man}). \end{array}$$

However, one would still have to identify and minimize connected components.

Note that the property and class hierarchies in our example are trees. So the attribute concepts of $\mathbb{K}$ are closed under suprema, and each concept label can be expressed by a single attribute. The suprema of properties may be of type "Resource". Such arcs are removed from the patterns; it can be shown (using the product property) that the resulting patterns can still be considered MSQs.

4 Connectedness

We have identified two components of the RDF graph in Fig. 1, and this corresponds to our understanding that objects are "connected" if they contribute to the same instance of a situation. The notion of connectedness deserves closer examination, however. Let us first observe that the notion of strong connectivity is not the right one, because the second to top concepts in Fig. 6 (we might name them "Lover" and "Beloved") would not have been created under this notion. But also, we can in general not rely on weak connectivity alone. If it had been stated explicitly that all persons are of type "Person", all persons would be connected through that class node.

Clearly, objects do not contribute to the same situation just on the basis of belonging to the same class, or having equal property values, like being of the same age. So connectedness of patterns should not rely on properties, classes or values. In this paper, no further requirements have been made, but if Liam liked lobscouse (a traditional sailors' dish), this would have not been sufficient because all objects would be connected through the lobscouse node. From a machine perspective, there is no indication in the RDF data (or the schema) that "ex:lobscouse" is of a different quality than "ex:Tom" or "ex:Mary". A system that generates patterns from automatically acquired data without proper preprocessing would possibly generate a large number of meaningless (and unnecessarily large) concepts because of such insubstantial connections, unless it can be guaranteed that some standard for making such distinctions is applied on the level of the ontology language.

Further questions may include if and how real-world objects should be connected through objects of spoken language; on what basis we could allow a pattern like the one describing the set of all wifes older than their husband, and at the same time keep meaningless comparison of values from having an impact on concept formation; or if pattern connection should be described in terms of edges rather than nodes, and if we can classify or characterize the kind of relations which contribute to pattern formation. It seems that, when dealing with pattern concepts, questions of philosophical ontology may have (at least subjectively) measurable impact through the notion of connectedness.

5 Related Work

In Sect. 2, the category theoretical framework for lattices of pattern concepts has been applied to triple graphs, and we will now show how it is applied to *simple concept graphs*. In [15], a simple concept graph over a power context family $\vec{\mathbb{K}} = (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2, \dots)$ is defined as a 5-tuple $(V, E, \nu, \kappa, \rho)$. We may interpret the 4-tuple (V, E, ν, κ) as a query, $\vec{\mathbb{K}}$ as a data source and the map ρ as a set of solutions. The map κ assigns concepts of the contexts $\mathbb{K}_i$ to the vertices and edges in V and E, respectively. The definition of queries should be independent from any particular data source, so it makes sense to interpret $\vec{\mathbb{K}}$ as a power context family describing schema information (background knowledge) instead, the contexts could describe attribute implications, like the one in Fig. 4. The map ρ will however not be part of the query. For the rest of the exposition, we shall call the 4-tuple (V, E, ν, κ) an *abstract concept graph* over $\vec{\mathbb{K}}$, after a term used in the first paper on concept graphs [14, p.300].

In [15], the triple (V, E, ν) is called a *relational graph*. The morphisms in the category of abstract concept graphs over $\vec{\mathbb{K}}$ are relational graph morphisms (which replaces condition (1) for triple graphs) satisfying (2). The product is defined in analogy to (3) (cartesian products of edges have to be taken individually for each arity k).

A datasource for the schema $\vec{\mathbb{K}}$ is a power context family $\vec{\mathbb{D}} = (\mathbb{D}_0, \mathbb{D}_1, \mathbb{D}_2, \dots)$ for which the attribute implications described by $\mathbb{K}_i =: (H_i, N_i, J_i)$ are satisfied in $\mathbb{D}_i =: (G_i, M_i, I_i)$, $i = 0, 1, 2, \dots$, and $N_i \subseteq M_i$ holds. We can represent $\vec{\mathbb{D}}$ by an abstract concept graph $\mathfrak{D} = (G_0, \bigcup_{i \geq 1} G_i, \mathrm{id}, \gamma_{\mathfrak{D}})$ with $\gamma_{\mathfrak{D}}(u) := ((u^{I_k} \cap$

$N_k)^{J_k}, u^{I_k} \cap N_k) \in \underline{\mathcal{B}}(\mathbb{K}_k)$ for $u \in G_k$. Let us furthermore define $\mathrm{Ext}_{\mathfrak{D}}(\kappa(u)) := \mathrm{Ext}(\kappa(u))^{J_k I_k}$ for $u \in V \cup E$. If we define a realization of $\mathfrak{G} =: (V, E, \kappa, \nu)$ over $\mathfrak{D}$ as a map with $\rho(u) \in \mathrm{Ext}_{\Delta}(\kappa(u))$ for all $u \in V \cup E$ and $\rho(e) = (\rho(v_1), \ldots, \rho(v_k))$, we obtain that the realizations ρ of $\mathfrak{G}$ over $\mathfrak{D}$ are precisely the morphisms $\rho : \mathfrak{G} \to \mathfrak{D}$. This means that product patterns for abstract concept graphs can be interpreted as MSQs, as was mentioned at the end of Sect. 1.

Triple graphs are now obtained as a special case of concept graphs: We define a power context family $\vec{\mathbb{D}}$ where $\mathbb{D}_0$ is a supercontext of $\mathbb{K}_0$ which in addition contains all objects (having only the "Resource" attribute), and where $\mathbb{D}_3 = (T, \emptyset, \emptyset)$ represents the set of triples.

In Relational Concept Analysis, concept lattices for different kinds of interrelated objects are generated by an iterative algorithm and, as in Fig. 6, illustrations displaying graphs of concepts have been given [4, 10]. Computational aspects of generating pattern structures are covered e.g. in [13, 12]. In [3], concept lattices are generated from Conceptual Graphs, but the approach seems to be tailored towards a more specific case where edges (and thus paths) express dependencies. A comparison with Relational Semantic Systems [16] still has to be made.

6 Conclusion

In [11], a lattice of closures of database queries, which links products of query patterns to the join operation on result tables, has been introduced. The queries and database were formalized by relational structures. The fact that the method could be applied again, first to RDF graphs and then to abstract concept graphs, suggests that the underlying category theoretical notions provide a recipe that could be applied to still different formalizations of queries, allowing in every case the mathematical description of lattices of most specific queries. Combinatorial explosion is a computational problem that quickly turns up when computing the concept lattices. From a practical perspective, the lattices are useless if complexity problems can not be solved. However, in order to support data exploration, patterns must be understandable, thus also limited in size, and a solution to this problem may entail a solution to the problem of computational complexity.

In the section on connectedness, we have seen that ontological considerations stand in a direct relation to the quality of computed concepts. As a first consequence, although the idea of treating all kinds of resources in a homogeneous way seemed appealing, triple graphs must be replaced by some other formalization which reflects the importance of the distinction between instances and classes/properties. While such questions naturally arise in the setting of pattern concepts, they seem to have no obvious analogy for standard FCA, where intents are sets of attributes. Pattern concepts have thus the potential to further and substantiate a perspective on FCA as a branch of mathematical modeling, where the entities to be modeled are not "real world" systems but rather systems of concepts. Future work may be concerned with extensions of RDF/RDFS which support this perspective.

References

1. SPARQL 1.1 Query Language. Tech. rep., W3C (2013), `http://www.w3.org/TR/sparql11-query`
2. Adámek, J., Herrlich, H., Strecker, G.E.: Abstract and concrete categories : the joy of cats. Pure and applied mathematics, Wiley, New York (1990)
3. Andrews, S., Polovina, S.: A mapping from conceptual graphs to formal concept analysis. In: Andrews, S., Polovina, S., Hill, R., Akhgar, B. (eds.) Proceedings of ICCS 2011. LNCS, vol. 6828, pp. 63 – 76. Springer (2011)
4. Azmeh, Z., Huchard, M., Napoli, A., Hacene, M.R., Valtchev, P.: Querying relational concept lattices. In: Proc. of the 8th Intl. Conf. on Concept Lattices and their Applications (CLA'11). pp. 377–392 (2011)
5. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 Turtle. Tech. rep., W3C (2014), `http://www.w3.org/TR/turtle`
6. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. In: Proceedings of the ninth annual ACM symposium on Theory of computing. pp. 77–90. STOC '77, ACM, New York, NY, USA (1977)
7. Ferré, S.: Conceptual navigation in RDF graphs with SPARQL-like queries. In: Kwuida, L., Sertkaya, B. (eds.) Proceedings of ICFCA 2010. LNCS, vol. 5986, pp. 193–208. Springer, Heidelberg (2010)
8. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Proceedings of ICCS 2001. LNCS, vol. 2120, pp. 129–142. Springer (2001)
9. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
10. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Annals of Mathematics and Artificial Intelligence 49(1-4), 39–76 (2007)
11. Kötters, J.: Concept lattices of a relational structure. In: Pfeiffer, H.D., Ignatov, D.I., Poelmans, J., Gadiraju, N. (eds.) Proceedings of ICCS 2013. LNCS, vol. 7735, pp. 301–310. Springer (2013)
12. Kuznetsov, S.O.: Computing graph-based lattices from smallest projections. In: Wolff, K.E., Palchunov, D.E., Zagoruiko, N.G., Andelfinger, U. (eds.) KONT/KPP. Lecture Notes in Computer Science, vol. 6581, pp. 35–47. Springer (2007)
13. Kuznetsov, S.O.: Fitting pattern structures to knowledge discovery in big data. In: Formal Concept Analysis, 11th International Conference, ICFCA 2013, Dresden, Germany, May 21-24, 2013. Proceedings. pp. 254–266 (2013), `http://dx.doi.org/10.1007/978-3-642-38317-5_17`
14. Wille, R.: Conceptual graphs and formal concept analysis. In: Lukose, D., Delugach, H.S., Keeler, M., Searle, L., Sowa, J.F. (eds.) Proceedings of ICCS 1997, 5th International Conference on Conceptual Structures. LNCS, vol. 1257, pp. 290–303. Springer, Heidelberg (1997)
15. Wille, R.: Formal concept analysis and contextual logic. In: Hitzler, P., Schärfe, H. (eds.) Conceptual Structures in Practice, pp. 137–173. Chapman & Hall/CRC (2009)
16. Wolff, K.E.: Relational scaling in relational semantic systems. In: Rudolph, S., Dau, F., Kuznetsov, S.O. (eds.) Proceedings of ICCS 2009. LNCS, vol. 5662, pp. 307–320. Springer (2009)

Variability representation in product lines using concept lattices: feasibility study with descriptions from Wikipedia's product comparison matrices

Jessie Carbonnel, Marianne Huchard, and Alain Gutierrez

LIRMM, CNRS & Université de Montpellier, France
jessiecarbonnel@gmail.com
surname.lastname@lirmm.fr

Abstract. Several formalisms can be used to express variability in a product line. Product comparison matrix is a common and simple way to display variability of existing products from a same family, but they lack of formalisation. In this paper, we focus on concept lattices, another alternative already explored in several works to express variability. We first propose a method to translate a description from existing product comparison matrices into a concept lattice using Formal Concept Analysis. Then, we propose an approach to represent the case where a product family is described by other product families with interconnected lattices using Relational Concept Analysis. Because of the combinatorial aspect of these approaches, we evaluate the scalability of the produced structures. We show that a particular structure (AOC-poset) possesses interesting properties for the studies that we envision.

Keywords: Product lines, Formal Concept Analysis, Variability Representation.

1 Introduction

In product line engineering [5], several formalisms can be used to depict variability. Variability representation usually requires to take into account a large amount of data, and it is important to provide tools to help designers to represent and exploit them.

Among existing formalisms, the most common is Product Comparison Matrices (PCMs). PCMs describe product properties in a tabular form. It is a simple way to display features of products from a same family and to compare them. However, there is no existing format or good practices to design these PCMs. Therefore, cells in PCMs lack of formalisation and it is difficult to perform automatic and efficient processing or analysis on them [4, 17]. Feature Models (FMs) constitute an alternative to PCMs. FMs describe a set of existing features and constraints between them, and thus represent all possible configurations of products from a same family [6, 9, 11, 12]. They depict variability in a more formal

© 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors.
Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

way than PCMs, but, in their stardard form, FMs focus exclusively on features and do not specify if an existing product is associated with a configuration.

Formal Concept Analysis (FCA) and concept lattices have already been studied to express variability [13]. From a set of objects described by attributes, FCA computes subsets of objects that have common attributes and structures these subsets in a hierarchical way in concept lattices. Concept lattices can represent in a more formal way than PCMs informations related to variability. Through their structure, concept lattices highlight constraints between attributes like FMs, while keeping the relation between existing products of the family and the possible configurations. Besides, contrarily to FMs, which can have many various forms depending design choices, concept lattices represent variability in a canonical form. Moreover, FCA can be extended by Relational Concept Analysis (RCA) which permits to take into account relationships between objects of separate lattices and provide a set of interconnected lattices. This is useful to classify sets of products from different categories.

Concept lattices formal and structural aspects make them good candidates to apply automatic or manual processing including product comparison, research by attributes, partial visualisation around points of interest, or decision support. But the exponential growth of the size of concept lattices can make them difficult to exploit. In this paper, we will study the dimensions of these structures and try to find out if depicting product line variability with concept lattices provides structures that can be exploited from a perspective of size. For this, we will build in a first phase concept lattices from existing descriptions. Because there are abundant and focus on both products and features, we will extract descriptions from PCMs. Besides, we can find PCMs that possess a feature whose value domain corresponds to a set of products described by another PCM. It is a special case where a product family is described by another product family. In a second phase, we will model this case by interconnecting concept lattices obtained from the descriptions of these two PCMs using Relational Concept Analysis.

In this paper, we want to answer these questions: How can we represent the variability expressed by a PCM with a concept lattice? To what extent can we model the case where a product family is described by another product family with RCA? Can we efficiently exploit structures obtained with FCA and RCA with regard to their size?

The remainder of this paper is organised as follows. In Section 2, we will review Formal Concept Analysis and propose a way to build a concept lattice from a PCM's description. In Section 3, we will study how to represent the case where a product family is described by another product family with Relational Concept Analysis. Then, in Section 4, we will apply these two methods on Wikipedia's PCMs to get an order of magnitude of the obtained structure size. Section 5 discusses related work. Section 6 presents conclusion and future work.

2 Formal Concept Analysis and product comparison matrices

This section proposes an approach to represent with a concept lattice the variability originally expressed in a PCM.

2.1 Concept lattices and AOC-posets

Formal Concept Analysis [8] is a mathematical framework which structures a set of objects described by attributes and highlights the differences and similarities between these objects. FCA extracts from a formal context a set of concepts that forms a partial order provided with a lattice structure: the concept lattice.

A formal context is a 3-tuple (O, A, R) where O and A are two sets and $R \subseteq O \times A$ a binary relation. Elements from O are called *objects* and elements from A are called *attributes*. A pair from R states that *the object o possesses the attribute a*. Given a formal context $K = (O, A, R)$, a concept is a tuple (E, I) such that $E \subseteq O$ and $I \subseteq A$. It depicts a maximal set of objects that share a maximal set of common attributes. $E = \{o \in O | \forall a \in I, (o, a) \in R\}$ is the concept's extent and $I = \{a \in A | \forall o \in E, (o, a) \in R\}$ is the concept's intent. Given a formal context $K = (O, A, R)$ and two concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$ from K, $C_1 \leq C_2$ if and only if $E_1 \subseteq E_2$ and $I_2 \subseteq I_1$. C_1 is a subconcept of C_2 and C_2 is a superconcept of C_1. When we provide all the concepts from K with the specialisation order $\leq$, we obtain a lattice structure called a concept lattice.

We represent intent and extent of a concept in an optimised way by making elements appear only in the concept where they are introduced. Figure 7 represents a concept lattice having simplified intent and extent. We call object-concept and attribute-concept the concepts which introduce respectively at least an object or an attribute. In Figure 7, *Concept_7* and *Concept_2* introduce neither attributes nor objects: their simplified intent and extent are empty. If they are not necessary, we can choose to ignore these concepts. *Attribute-Object-Concept poset* (AOC-poset) from a formal context K is the sub-order of $(C_K, \leq)$ restricted to object-concepts and attribute-concepts. Figure 4 presents the AOC-poset matching the concept lattice from Figure 7. In our case, interesting properties of concept lattices with regard to variability are preserved in AOC-posets: this smaller structure can be used as an alternative to concept lattices.

2.2 Product Comparison Matrix

A PCM describes a set of products from a same family with variable features in a tabular way. Figure 1 presents a PCM which describes four products against two features, taken from Wikipedia.

We notice that the cells of this PCM lack of formalisation: in this, different values have the same meaning (*Object-Oriented* and *OO*) and it seems that there are no rules on the use of value separator (',' or '&' or 'and').

Language	Standardized	Paradigm
Java	Yes	Functional, Imperative, OO
Perl	No	Functional & Procedural & Imperative
Php	No	Functional, Procedural, Imperative and Object-Oriented
C#	Yes	functional, procedural, imperative, Object Oriented

Fig. 1. Excerpt of a Product Comparison Matrix on programming languages (http://en.wikipedia.org/wiki/Comparison_of_programming_languages, July 2014)

2.3 Processing PCMs to get formal context

If we want to automatically build concept lattices from this kind of descriptions, we need to make the cell values respect a format in order to extract and process them. In [17], authors identify height different types of cell values:

– *yes/no values* that indicate if the criterion is satisfied or not,
– *constrained yes/no values* when the criterion is satisfied under conditions,
– *single-value* when the criterion is satisfied using this value,
– *multi-values* when several values can satisfy the criterion,
– *unknown value* when we do not know if the criterion is satisfied,
– *empty cell*,
– *inconsistent value* when the value is not related to the criterion,
– *extra information* when the cell value offers additional informations.

We clean each type of cells as follows. Empty cells are not a problem, even though they indicate a lack of information. Inconsistent values should be detected, then clarified or erased. Unknown values and extra informations will be simply erased. Other types of cells could have either one single value or a list of values. We will always use a coma as value separator. Values with the same meanings will be written in the same way. Once we have applied these rules on a PCM, we consider that this PCM is cleaned. A cleaned PCM can own three types of features:

– *simple boolean*,
– *constrained boolean*,
– *non-boolean*.

Since automatic process is difficult on original PCMs, we clean them manually. When we clean the PCM in Figure 1, we obtain the PCM in Figure 2.

We can now automatically extract values from cleaned PCMs to generate *formal contexts*. Given a set of objects and a set of binary attributes, a formal context is a binary relation that states which attributes are possessed by each object. In summary, we want to convert a set of multivalued features (PCM) into a set of binary attributes (formal context).

Scaling technique [8] consists in creating a binary attribute for each value (or group of values) of a multivalued feature. Boolean features can produce a single attribute to indicate if either or not the object owns this feature. Yet

Language	Standardized	Paradigm
Java	Yes	Functional, Imperative, Object-Oriented
Perl	No	Functional, Procedural, Imperative
Php	No	Functional, Procedural, Imperative, Object-Oriented
C#	Yes	Functional, Procedural, Imperative, Object-Oriented

Fig. 2. PCM in Figure 1 cleaned manually

because of empty cells, the fact that an object does not possess an attribute could also mean that the cell from the PCM was left blank. To be more precise, we can choose to generate two attributes: one to indicate that the object possesses the feature, and one to indicate that the object does not possess the feature. Constrained boolean features can be processed in the same way than simple boolean features by producing one or two attributes, with the difference that we can keep constraints in the form of attributes. Non-boolean features will simply produce an attribute per value or group of values. We applied scaling technique on the cleaned PCM of Figure 2 and got the formal context in Figure 3.

Language	Standardized:Yes	Standardized:No	Functional	Procedural	Imperative	Object -Oriented
Java	x		x		x	x
Perl		x	x	x	x	
Php		x	x	x	x	x
C#	x		x	x	x	x

Fig. 3. Formal context obtained after scaling the cleaned PCM in Figure 2

The structure of concept lattices and AOC-posets permits to highlight interesting properties from variability point of view: they classify objects depending on their attributes and emphasise relations between these attributes (e.g. require, exclude). For instance: attributes introduced in the top concept are owned by all objects; attributes which are introduced in the same concept always appear together; if an object o_1 is introduced in a sub-concept of a concept introducing an object o_2, o_1 possesses all the attributes of o_2 and other attributes; two objects introduced in the same concept possess the same attributes. Feature models show part of this information, mainly reduced to relations between attributes, as they do not include the products in the representation.

Figure 7 represents the concept lattice from the formal context of Figure 3. Figure 4 represents the AOC-poset from the formal context of Figure 3. In these two structures, we can see that: all languages permit to write programs according to functional and imperative paradigms; the product *Php* has all the attributes of *Perl* in addition to the attribute *Object Oriented*; all standardised languages are object-oriented.

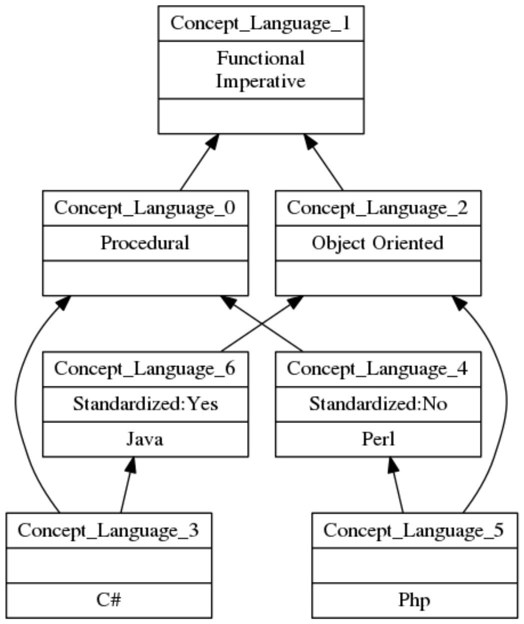


Fig. 4. AOC-poset from the formal context of Figure 3

3 Relational Concept Analysis and interconnected product lattices

This section proposes an approach to model the case where a PCM possesses a feature whose value domain corresponds to a set of products described by another PCM.

3.1 Modeling interconnected families of products with RCA

We illustrate the modeling of interconnected families of products with an extension of our example. We can find on Wikipedia a PCM on *Wikisoftwares* that refers to *Programming Languages*: we want to structure wikisoftwares according to programming languages in which they are written. We assume a PCM about wikisoftwares that owns a boolean feature *Open Source* and a constrained boolean feature *Spam Prevention*. We applied Section 2 approach and obtained the formal (objects-attributes) context in Figure 5. Figure 8 presents the concept lattice associated with the context in Figure 5.

Wikisoftware	OS:Yes	OS:No	SP:Yes	SP:No	SP:Captcha	SP:Blacklist
TeamPage		x	x		x	
Socialtext			x			
MindTouch		x	x			
DokuWiki	x		x			x
EditMe		x	x		x	

Fig. 5. Objects-attributes context of Wikisoftwares

Relational Concept Analysis (RCA) [10] extends FCA to take into account relations between several sets of objects. Each set of objects is defined by its own attributes (in an objects-attributes context) and can be linked with other sets of objects. A relation between two sets of objects is stated by a relational context (objects-objects context). A relational context is a 3-tuple (O_1, O_2, I) where O_1 (source) and O_2 (target) are two sets of objects such that there are two formal contexts (O_1, A_1, R_1) and (O_2, A_2, R_2), and where $I \subseteq O_1 \times O_2$ is a binary relation.

We want to express the relation *isWritten* between objects of *Wikisoftwares* and objects of *Programming languages*. TeamPage and EditMe are written in Java, SocialText in Perl, DokuWiki in Php and Mindtouch in both Php and C#. We link each wikisoftware with corresponding programming languages in an objects-objects context, and we present it in Figure 6.

isWritten	Java	Perl	Php	C#
TeamPage	x			
Socialtext		x		
MindTouch			x	x
DokuWiki			x	
EditMe	x			

Fig. 6. Objects-objects context expressing the relation between objects of Wikisoftwares and Programming languages

3.2 Processing interconnected families of products

Given an objects-objects context $R_j = (O_k, O_l, I_j)$, there are different ways for an object from O_k domain to be in relation with a concept from O_l. For instance: an object is linked (by I_j) to at least one object of a concept's extent (*existential*

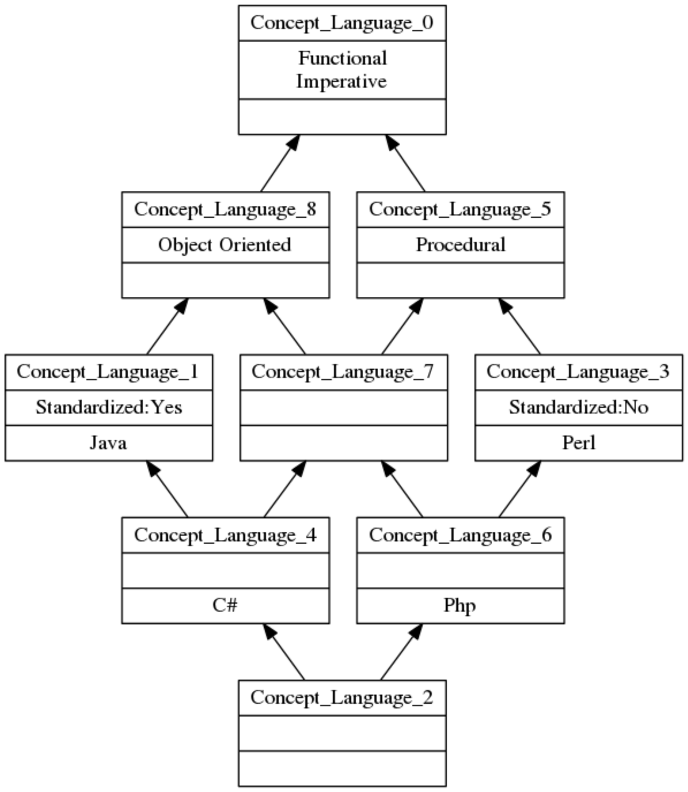


Fig. 7. Concept lattice of Programming languages, step 0 (built with RCAExplore: http://dolques.free.fr/rcaexplore.php)

scaling); an object is linked (by I_j) only to objects of a concept's extent (*universal scaling*). For each relation of R, we specify which scaling operator is used.

In RCA, objects-attributes contexts are extended according to objects-objects contexts to take into account relations between objects of different sets. For each objects-objects context $R_j = (O_k, O_l, I_j)$, RCA extends the objects-attributes context of the set of objects O_k by adding *relational attributes* according to concepts of the lattice associated with the objects-attributes O_l. Each concept c from O_l gives a relational attribute q r :c where q is a scaling operator and r is the relation between O_k and O_l. A relational attribute appears in a lattice just as the other attributes, with the difference that it can be considered like a *reference* to a concept from another lattice.

As shown on the example, data are represented in a Relational Context Family (RCF), which is a tuple (K, R) such that K is a set of objects-attributes contexts $K_i = (O_i, A_i, I_i)$, $1 \leq i \leq n$ and R is a set of objects-objects contexts

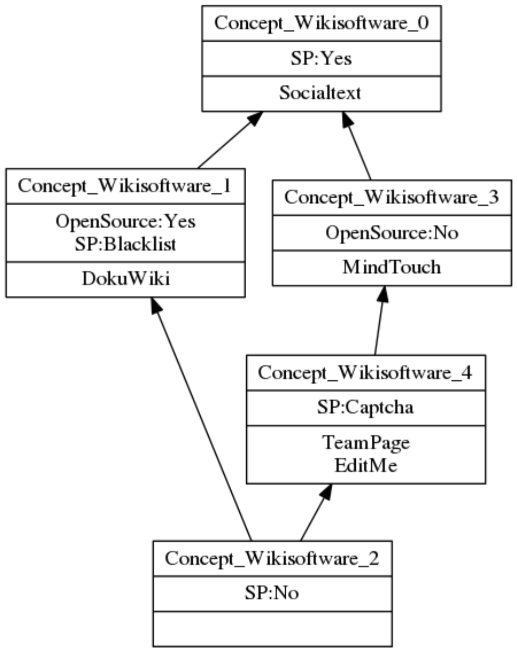


Fig. 8. Concept lattice of Wikisoftwares, step 0

$R_j = (O_k, O_l, I_j)$, $1 \leq j \leq m$, with $I_j \subseteq O_k \times O_l$. Given an objects-attributes context $\mathcal{K} = (O, A, I)$, we define $rel(\mathcal{K})$ the set of relations (objects-objects contexts) of R which have O for domain, and ρ a function which associates a scaling operator to each objects-objects context of R. For each step, we extend the context $\mathcal{K}$ by adding relational attributes from each context of $rel(\mathcal{K})$: we obtain the *complete relational extension of* $\mathcal{K}$. When we compute the complete relational extension of each context of K, we obtain the *complete relational extension of the RCF*.

RCA generates a succession of contexts and lattices associated with the RCF (K, R) and ρ. In step 0, RCA generates lattices associated with contexts of K. $K^0 = K$. In step $e + 1$, RCA computes complete relational extension of the K^e contexts. The obtained extended contexts (K^{e+1}) possess relational attributes which refer to concepts of lattices obtained in the previous step.

In our example, the RCF is composed of *Wikisoftware*, *Programming Language* and of the objects-objects context *isWritten*. When we compute the complete relational extension of this RCF, we extend the objects-attributes context of *Wikisoftware* with relational attributes which refer to each concept of *Programming language*. Figure 7 and Figure 8 represent lattices of *Wikisoftware* and *Programming language* at step 0. Figure 9 presents the concept lattice from the extended objects-attributes context of *Wikisoftwares*, at step 1. In this example, we cannot go further than step 1.

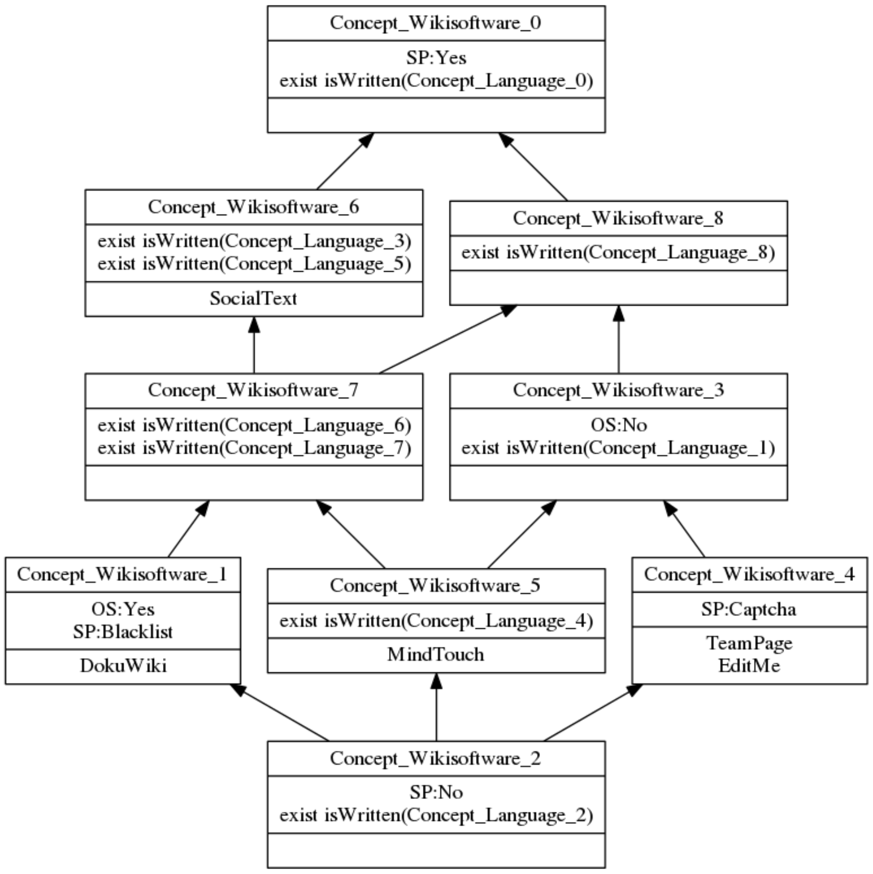

Concept_Wikisoftware_0
SP:Yes
exist isWritten(Concept_Language_0)
Concept_Wikisoftware_6
exist isWritten(Concept_Language_3)
exist isWritten(Concept_Language_5)
SocialText
Concept_Wikisoftware_8
exist isWritten(Concept_Language_8)
Concept_Wikisoftware_7
exist isWritten(Concept_Language_6)
exist isWritten(Concept_Language_7)
Concept_Wikisoftware_3
OS:No
exist isWritten(Concept_Language_1)
Concept_Wikisoftware_1
OS:Yes
SP:Blacklist
DokuWiki
Concept_Wikisoftware_5
exist isWritten(Concept_Language_4)
MindTouch
Concept_Wikisoftware_4
SP:Captcha
TeamPage
EditMe
Concept_Wikisoftware_2
SP:No
exist isWritten(Concept_Language_2)

Fig. 9. Concept lattice of Wikisoftwares, step 1

In Figure 9, relational attributes are read like references to concepts in the lattice of *Programming languages* at step 0 (Figure 7). An extended concept lattice gives us the same kind of informations that are emphasised in a basic concept lattice, but it takes into account attributes from other product families. This brings a new dimension to research and classification of products from a same family. In our example, it permits us to select a wikisoftware depending on the paradigm of its programming language. In Figure 9, we can read for instance:

- *DokuWiki* (concept 1) is written in a programming language characterised by concepts 6, 7, 8, 3, 5 and 0 of *Programming languages*, corresponding

to attributes *Standardized:No*, *Procedural*, *Functional*, *Object Oriented* and *Imperative*;
- *Team Page* and *EditMe* are equivalent products because they are introduced in the same concept (same attributes and same relational attributes);
- a wikisoftware not *Open Source* is written in a standardised language;
- an *Open Source* wikisoftware is written in an unstandardised language.

4 Assessing scalability of the approach

Until now, we proposed a first method to obtain a concept lattice from a PCM's description and a second method to depict the case where features possess their own PCM with interconnected lattices. In this section, we evaluate these two methods on existing PCMs from Wikipedia to get an order of magnitude of the obtained structures size.

The number of generated concepts from a formal context depends on the number of objects, the number of attributes and the form of the context: this number can reach $2^{min(|O|,|A|)}$ with a lattice, and $|O| + |A|$ for an AOC-poset.

In the following tests, we generate both concept lattices and AOC-posets to emphasise the impact of deleting concepts which introduce neither attributes nor objects on the size of the structure. Each test was made twice, a first time with *full formal contexts* (scaling technique giving two attributes for each boolean feature, and keeping constraints in the form of attributes for constrained boolean features) and a second time with *reduced fomal contexts* (scaling technique giving one attribute for each boolean feature).

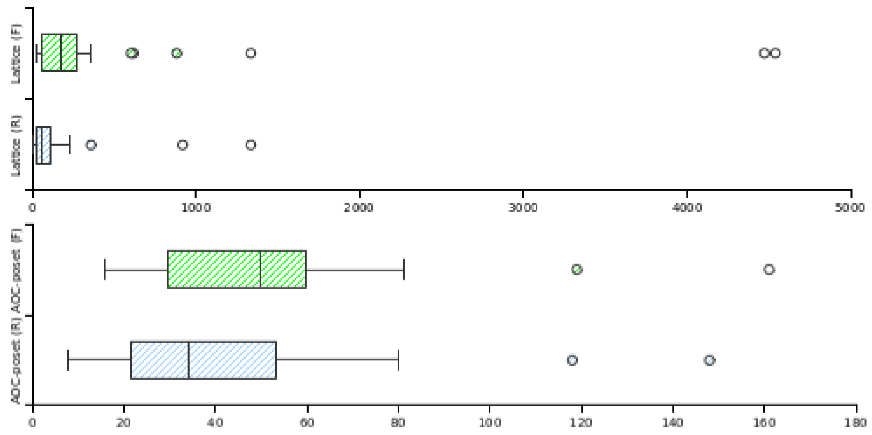
4.1 Scalability for non-connected PCMs (FCA)

Firstly, we analysed 40 PCMs from Wikipedia without taking into account relations between products. These 40 PCMs were converted into formal contexts with the method of Section 2. Results are presented in Figure 10. We have analysed 1438 products, generated 4639 binary attributes and 26002 formal concepts.

Most of the concept lattices possess between 50 and 300 concepts, but some of them can reach about 5000 concepts: this number is very high, and it would be difficult to quickly process these data. Reduced contexts (blue plots) give smaller structures, but some of them remain considerable. Thus, results of AOC-posets are encouraging: the highest number of concepts obtained is 161. Most of them possess between 30 and 60 concepts.

4.2 Scalability for connected PCMs (RCA)

Secondly, we made the same type of tests on structures which have been extended with a relation, using method of Section 3. We wanted to realise these tests on a quite important number of relationships. Yet, it is simple to find PCMs on Wikipedia but it is more difficult to automatically find relationships between these PCMs. To simulate relations between PCMs we used in the first test,



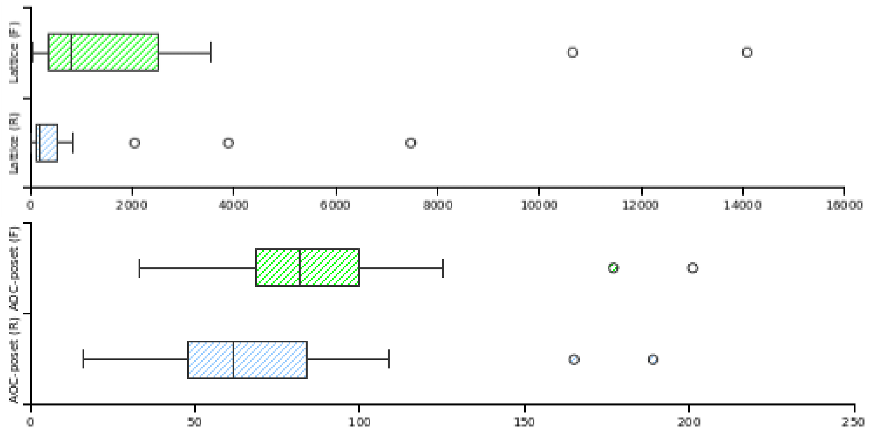
	Lattices (F)	Lattices (R)	AOC-posets (F)	AOC-posets (R)
Mean	427.375	131.35	50.3	41.025
Median	174	62.5	50	34
Minimum	21	8	16	8
Maximum	4551	1336	161	148
First Quartile	49	28.25	29.25	20.5
Third Quartile	282.5	127.75	61.25	53.75

Fig. 10. Box plots on the number of concepts obtained with concept lattices (top) and AOC-poset (bottom), from full formal contexts (green) and reduced formal contexts (blue)

we choose to automatically generate random objects-objects contexts based on two existing relations we found on Wikipedia. We analysed these relations and found out that about 75% of objects are linked to at least another object and that 95% of linked objects are linked to a single other object. We formed 20 pairs of objects-attributes contexts and generated object-objects contexts for each pair according to our analysis.

Results are presented in Figure 11. Concept lattices are huge (some can reach 14000 concepts) whereas AOC-poset remain relatively affordable (about 200 concepts).

These results match with the products used in a very simplified form for illustrating the approach. In real data, the first existing relation is between *LinuxDistribution* (77 objects, 25 features) and *FileSystem* (103 objects, 60 features). With a lattice, we obtain 1174 concepts (full context) and 1054 concepts (reduced context). With an AOC-poset, we obtain 188 then 179 concepts. The second relation is between *Wikisoftware* (43 objects, 12 features) and *Programming Language* (90 objects, 5 features). With a lattice, we obtain 282 concepts (full context) and 273 concepts (reduced context). With an AOC-poset, we obtain 87 and then 86 concepts. All the datasets (extracted from



	Lattices (F)	Lattices (R)	AOC-posets (F)	AOC-posets (R)
Mean	2263.5	878.2	89.95	72.55
Median	793	194	82	61.5
Minimum	61	24	33	16
Maximum	14083	7478	201	189
First Quartile	359.25	108.25	67.5	47.25
Third Quartile	2719.5	597.75	101.25	87.25

Fig. 11. Box plots on the number of concepts obtained with formal contexts extended with RCA

wikipedia and generated) are available for reproducibility purpose at: `http://www.lirmm.fr/recherche/equipes/marel/datasets/fca-and-pcm`.

According to these results, we can deduce that the concept lattices obtained possess a majority of empty concepts (which introduce neither attributes nor objects): AOC-poset appears like a good alternative to generate less complex structures, and keeps variability informations in the same way that concept lattices. These results on product description datasets, that are either real datasets, or datasets generated with respect to existing real one profile, allow us to think that is is realistic to envision using FCA and RCA for variability representation within a canonical form integrating features and products.

5 Related work

To our knowledge, the first research work using Formal Concept Analysis to analyse relationships between product configurations and variable features to assist construction of a product line has been realised in Loesh and Ploederer [13]. In this approach, the concept lattice is analysed to extract informations about groups of features always present, never present, always present together or never present together. Authors use these informations to describe constraints between features and propose restructurations of these features (merge, removal or identification of alternatives feature groups). In [14], authors study, in an aspect-oriented requirements engineering context, a concept lattice which classifies scenarios by functional requirements. They analyse on the one hand the relation between concepts and quality requirements (usability, maintenance), and on the other hand interferences between quality requirements. Also, they analyse the impact of modifications. This analysis has been extended by observations about product informations contained in the lattice (or the AOC-poset), and on the manner that some work implicitly use FCA, without mentioning it [1]. In the present work, we show how to extend the original approach, which analyses products described by features, to a more general case where there are features that are products themselves. Moreover, we evaluate the scale up of FCA and RCA on product families described by PCMs from Wikipedia and linked by relationships randomly generated.

In the domain of product lines, another category of works is interested in identification of features in source code using FCA [19, 3]. In this case, described entities are variants of software systems which are described by source code and authors try to produce groups of source code elements that can be candidates to be features. Some works search for traceability links between features and the code [16]. In [7], authors cross-reference source code parts and scenarios which execute them and use features. The goal is to identify parts of the code which correspond to feature implementation. In our case, we do not work on source code, nor with scenarios, but with existing product descriptions.

Finally, a last category of works study feature organisation in FM with FCA. Some approaches [20, 15] use conceptual structures (concept lattice or AOC-poset) to recognise contraints, but also to suggest sub-features typed relations linked to the domain. In the article [15], authors study in detail the extraction of implication rules in the lattice and cover relationships, to determine for instance if a set of features covers all the products. Recent works [2, 18] focus on emphasise logical relationships between features in a FM and on the identification of transverse constraints. These logical relationships are more specifically used in [18] to analyse the variability of a set of architectural configurations. In the present work, we generate a structure or several interconnected structures. These structures are created to analyse variability, but we do not consider issues about FM construction.

6 Conclusion

In this paper, we analyse the feasibility of using Formal Concept Analysis and Concept Lattices as a complement to Product Comparison Matrices to represent variability in product lines. We propose a method to convert a description from a product comparison matrix to a concept lattice using the scaling technique. We also propose a way to model the special case where a product family is described by another product family with Relational Concept Analysis. We obtain interconnected lattices that bring a new dimension to research and classification of products when they are in relation with other product families.

Subsequently, we realise series of tests to determine an order of magnitude of the number of concepts composing the structures obtained firstly with FCA by converting PCMs into formal contexts, and secondly with RCA by introducing relations between these contexts. In these tests, we compare two structures: concept lattices which establish a sub-order among all the concepts and AOC-posets which establish a sub-order among the concepts which introduce at least an attribute or an object. It seems that most of the concepts do not introduce any information and AOC-poset appears like a more advantageous alternative, in particular for presenting information to an end-user. Concept lattices are useful too, when they are medium-size, and for automatic data manipulations. We also show the effect of two different encoding of boolean values.

In the future, we will use the work of [4] to automatise as much as possible the conversion of PCMs. Moreover, researches on the classification process (using different scaling operators) and its applications on variability will be performed to complete this analysis. Also, a detailed study of the possibilities offered by RCA to model other cases is considered. Finally, it could be interesting to study transitions between different structures like FMs, PCMs, AOC-posets and concept lattices to be able to select one depending on the kind of operation we want to apply.

References

1. Al-Msie 'deen, R., Seriai, A.D., Huchard, M., Urtado, C., Vauttier, S., Al-Khlifat, A.: Concept lattices: A representation space to structure software variability. In: ICICS 2014: The fifth International Conference on Information and Communication Systems. pp. 1 – 6. Irbid, Jordan (Apr 2014), http://hal-lirmm.ccsd.cnrs.fr/lirmm-01075533
2. Al-Msie'deen, R., Huchard, M., Seriai, A., Urtado, C., Vauttier, S.: Reverse engineering feature models from software configurations using formal concept analysis. In: Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014. pp. 95–106 (2014)
3. Al-Msie'deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S., Salman, H.E.: Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing. In: Proc. of The 25th SEKE. pp. 244–249 (2013)
4. Bécan, G., Sannier, N., Acher, M., Barais, O., Blouin, A., Baudry, B.: Automating the formalization of product comparison matrices. In: Proc. of the 29th ACM/IEEE ASE '14. pp. 433–444 (2014)
5. Clements, P.C., Northrop, L.M.: Software product lines: practices and patterns. Addison-Wesley (2001)
6. Czarnecki, K., Eisenecker, U.W.: Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
7. Eisenbarth, T., Koschke, R., Simon, D.: Locating features in source code. IEEE Trans. Softw. Eng. 29(3), 210–224 (2003)

8. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
9. Griss, M.L., Favaro, J., Alessandro, M.d.: Integrating Feature Modeling with the RSEB. In: Proceedings of the 5th International Conference on Software Reuse. pp. 76–. ICSR '98, IEEE Computer Society, Washington, DC, USA (1998), `http://dl.acm.org/citation.cfm?id=551789.853486`
10. Hacène, M.R., Huchard, M., Napoli, A., Valtchev, P.: Relational concept analysis: mining concept lattices from multi-relational data. Ann. Math. Artif. Intell. 67(1), 81–108 (2013)
11. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study (November 1990)
12. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: Form: A feature-oriented reuse method with domain-specific reference architectures. Ann. Software Eng. 5, 143–168 (1998)
13. Loesch, F., Ploedereder, E.: Restructuring variability in software product lines using concept analysis of product configurations. In: Proc. of the 11th IEEE ECSMR. pp. 159–170 (2007)
14. Niu, N., Easterbrook, S.M.: Concept analysis for product line requirements. In: Proc. of the 8th AOSD 2009. pp. 137–148 (2009)
15. Ryssel, U., Ploennigs, J., Kabitzsch, K.: Extraction of feature models from formal contexts. In: Proc. of ACM SPLC '11. pp. 4:1–4:8 (2011)
16. Salman, H.E., Seriai, A., Dony, C.: Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In: Proc. of the14th IEEE IRI. pp. 209–216 (2013)
17. Sannier, N., Acher, M., Baudry, B.: From comparison matrix to variability model: The wikipedia case study. In: Proc. of the 28th IEEE/ACM ASE 2013. pp. 580–585 (2013)
18. Shatnawi, A., Seriai, A.D., Sahraoui, H.: Recovering architectural variability of a family of product variants. In: To appear in Proc. of the 14th ICSR (2015)
19. Xue, Y., Xing, Z., Jarzabek, S.: Feature location in a collection of product variants. In: Proc. of the 19th IEEE WCRE. pp. 145–154 (2012)
20. Yang, Y., Peng, X., Zhao, W.: Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In: Proc. of the 16th IEEE WCRE. pp. 215–224 (2009)