A new perspective on the Close-by-One algorithm

Radek Janoštík

Author Paper of Dissertation Thesis

Department of Computer Science
Faculty of Science
Palacký University Olomouc
2021

Uchazeč
Mgr. Radek Janoštík
radek.janostik@gmail.com

Školitel
doc. RNDr. Jan Konečný, Ph.D.

Místo a termín obhajoby

Oponenti

S disertační prací a posudky se bude možné seznámit na katedře informatiky PřF UP, 17. listopadu 12, 771 46 Olomouc.

Abstract – The Close-by-One (CbO) algorithm is a well-known algorithm used in formal concept analysis (FCA). We shed a new light on CbO: First, we propose and evaluate a novel algorithm for computation of the Duquenne-Guigues basis which combines CbO and LinClosure algorithms. This combination enables us to reuse attribute counters used in LinClosure and speed up the computation. Second, we describe LCM, an algorithm for enumeration of frequent closed itemsets in transaction databases, in terms of FCA and show that LCM is basically the CbO algorithm with multiple speed-up features for processing sparse data. Third, we show that FCA and Logical Analysis of Data (LAD) utilize the same basic building blocks, which enable us to develop an interface between the two methodologies. We provide some preliminary benefits of the interface; most notably efficient algorithms for computing spanned patterns in LAD using algorithms of FCA.

Preface

Computation of all closed sets of a closure operator is an important task in computer science as it is used in many fields: boolean factor analysis [18], data mining [92] and databases [70], to name just a few.

Closed sets also play a crucial role in formal concept analysis [36], its basic notions—extents and intents—are also closed sets. Many algorithms for computation of all closed sets have been designed [67]. One of the most efficient is the Close-By-One (CbO) algorithm [62] from which many variants were developed, namely, FCbO [73] and the family of In-Close algorithms [5, 6, 7, 8, 9].

This thesis focuses on a new view of the CbO algorithm. We bring our three important results. Firstly, its variant – LinCbO, our new algorithm for computation of the Duquenne-Guigues basis, can reuse values of LinClosure's attribute counters during a computation, which, dramatically speeds up a computation. Secondly, the well-known algorithm LCM [86, 87] in the data mining community is basically CbO with some speed-up features. Finally, thanks to the introduced interface between formal concept analysis and logical analysis of data [1], the CbO algorithm can be used in the logical analysis of data.

Particular parts of this document are based on our following work:

[50] Radek Janostik, Jan Konecny and Petr Krajča. LinCbO: fast algorithm for computation of the Duquenne-Guigues basis. *CoRR*, abs/2011.04928, 2020. *(submitted to Information Sciences, currently in review)*

[47] Radek Janostik, Jan Konecny and Petr Krajča. LCM is well implemented CbO: study of LCM from FCA point of view. In *CLA*, pages 47–58, 2020.

[48] Radek Janostik, Jan Konecny and Petr Krajča. Interface between Logical Analysis of Data and Formal Concept Analysis. *European Journal of Operational Research*, 2020.

We also published the article

[46] Radek Janostik and Jan Konecny. General framework for consistencies in decision contexts. *Information Sciences*, 530:180–200, 2020.

the topic of which is quite different from the others. Therefore, it was omitted from this document.

This paper is organized as follows. In Section 1 we provide preliminaries which covers basic notions needed for understanding the whole author paper.

In Section 2 we introduce our new algorithm for computing the Duquenne-Guigues basis called LinCbO. It is based on the CbO algorithm with the LinClosure algorithm as closure operator. It reuses values of counters from the previous calls of the closure.

Section 3 provides a description of the LCM algorithm from the FCA point of view. We show that it is basically the CbO with interesting features which were hidden in the implementation. We describe them in detail without delving into implementation details.

In Section 4 we describe the interface between two methodologies: formal concept analysis and logical analysis of data. Thanks to the interface we can use algorithms from FCA for solving some problems from LAD.

Finally, in Section 5 we present the conclusions of this document.

Thesis was supported by the grants:

- IGA UP 2020 of Palacký University Olomouc, No. IGA_PrF_2020_019,
- JG 2019 of Palacký University Olomouc, No. JG_2019_008.

1 Preliminaries

In this section, we recall notions used in the rest of the author paper. Only necessary foundations are presented; more details can be found in the cited sources.

1.1 Closure operators

A *closure system* in a set Y is any system $\mathcal{S}$ of subsets of Y which contains Y and is closed under arbitrary intersections.

A *closure operator* on a set Y is a mapping $c : 2^Y \to 2^Y$ satisfying for each $A, A_1, A_2 \subseteq Y$:

$$A \subseteq c(A) \tag{1}$$
$$A_1 \subseteq A_2 \quad \text{implies} \quad c(A_1) \subseteq c(A_2) \tag{2}$$
$$c(A) = c(c(A)). \tag{3}$$

The closure systems and closure operators are in one-to-one correspondence. Specifically, for a closure system $\mathcal{S}$ in Y, the mapping $c_{\mathcal{S}} : 2^Y \to 2^Y$ defined by

$$c_{\mathcal{S}}(A) = \bigcap\{B \in \mathcal{S} \mid A \subseteq B\}$$

is a closure operator. Conversely, for a closure operator c on Y, the set

$$\mathcal{S}_c = \{A \in 2^Y \mid c(A) = A\}$$

is a closure system. Furthermore, $\mathcal{S}_{c_{\mathcal{S}}} = \mathcal{S}$ and $c_{\mathcal{S}_c} = c$.

For further details please refer to [19].

	1	2	3	4	5
a	1	0	1	1	1
b	1	1	1	0	1
c	1	1	1	0	0
d	0	0	1	0	1

Figure 1: Example of formal context with objects $\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}$ and attributes $1, 2, 3, 4, 5$.

1.2 Formal concept analysis

Formal concept analysis (FCA) [36, 23] is a method of relational data analysis invented by Rudolf Wille [90]. It is based on a formalization of a certain philosophical view of conceptual knowledge [53]. FCA identifies interesting clusters (formal concepts) in a collection of objects and their attributes, and organizes them into a structure called a concept lattice.

It has been applied, for example, in software engineering [81, 44, 84], web mining [27, 28], organization of web search results [25, 24], text mining and linguistics [45], analysis of medical and biological data [15, 52, 51], and crime data [76, 74]. FCA has also been used in context machine learning. A model of learning from positive and negative examples called JSM-method has been described in terms of FCA [34, 59, 63].

1.2.1 One-valued (basic) setting

An input to FCA is a triplet $\langle X, Y, I\rangle$, called a *formal context*, where X, Y are non-empty sets of objects and attributes respectively, and I is a binary relation between X and Y. The presence of an object-attribute pair $\langle x, y\rangle$ in the relation I means that the object x has the attribute y.

Finite contexts are usually depicted as tables, in which rows represent objects in X, columns represent attributes in Y, ones in its entries mean that the corresponding object-attribute pair is in I; see Fig. 1 for an example.

The formal context $\langle X, Y, I\rangle$ induces so-called *concept-forming operators*:

> ${}^{\uparrow} : \mathbf{2}^X \to \mathbf{2}^Y$ assigns to a set A of objects the set $A^{\uparrow}$ of all attributes shared by all the objects in A.
>
> ${}^{\downarrow} : \mathbf{2}^Y \to \mathbf{2}^X$ assigns to a set B of attributes the set $B^{\downarrow}$ of all objects which share all the attributes in B.

Formally, for all $A \subseteq X, B \subseteq Y$ we have

$$A^{\uparrow} = \{y \in Y \mid \forall x \in A : \langle x, y\rangle \in I\},$$
$$B^{\downarrow} = \{x \in X \mid \forall y \in B : \langle x, y\rangle \in I\}.$$

For singletons, we use a shortened notation and write $x^{\uparrow}$, $y^{\downarrow}$ instead of $\{x\}^{\uparrow}$, $\{y\}^{\downarrow}$, respectively.

Fixed points of the concept-forming operators, i.e. pairs $\langle A, B\rangle \in \mathbf{2}^X \times \mathbf{2}^Y$ satisfying

$$A^{\uparrow} = B \quad \text{and} \quad B^{\downarrow} = A,$$

are called *(one-valued) formal concepts*. The sets A and B in a formal concept $\langle A, B\rangle$ are called the *extent* and the *intent*, respectively.

The set of all formal concepts in $\langle X, Y, I\rangle$ is denoted by $\mathcal{B}^{\uparrow\downarrow}(I)$. This set endowed with the order $\leq$, given by

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \quad \text{iff} \quad A_1 \subseteq A_2 \qquad \text{for all } \langle A_1, B_1\rangle, \langle A_2, B_2\rangle \in \mathcal{B}^{\uparrow\downarrow}(I),$$

forms a complete lattice called a *(one-valued) concept lattice*. We denote by $\mathrm{Ext}^{\uparrow\downarrow}(I)$ and $\mathrm{Int}^{\uparrow\downarrow}(I)$ the set of all extents and the set of all intents in $\mathcal{B}^{\uparrow\downarrow}(I)$, respectively; formally:

$$\mathrm{Ext}^{\uparrow\downarrow}(I) = \{A \mid \langle A, B\rangle \in \mathcal{B}^{\uparrow\downarrow}(I)\},$$
$$\mathrm{Int}^{\uparrow\downarrow}(I) = \{B \mid \langle A, B\rangle \in \mathcal{B}^{\uparrow\downarrow}(I)\}.$$

For each $A \subseteq X, B \subseteq Y$, we have

$$A \in \mathrm{Ext}^{\uparrow\downarrow}(I) \text{ iff } A = A^{\uparrow\downarrow} \text{ and } B \in \mathrm{Int}^{\uparrow\downarrow}(I) \text{ iff } B = B^{\downarrow\uparrow}.$$

For a formal context $\langle X, Y, I\rangle$, the set $\mathrm{Int}^{\uparrow\downarrow}(I)$ of its intents is a closure system. The corresponding closure operator, $c_{\mathrm{Int}^{\uparrow\downarrow}(I)}$, is equal to the composition $^{\downarrow\uparrow}$ of concept-forming operators.

1.2.2 Attribute implications, bases, Duquenne-Guigues basis and its computation

An *attribute implication* is an expression of the form $L \Rightarrow R$ where $L, R \subseteq Y$ are sets of attributes.

We say that $L \Rightarrow R$ is valid in a set of attributes $M \subseteq Y$ if

$$L \subseteq M \text{ implies } R \subseteq M.$$

The fact that $L \Rightarrow R$ is valid in M is written as $\|L \Rightarrow R\|_M = 1$.

We say that $L \Rightarrow R$ is valid in a context $\langle X, Y, I\rangle$ if it is valid in every object intent $x^{\uparrow}$, i.e.

$$\|L \Rightarrow R\|_{x^{\uparrow}} = 1 \qquad \forall x \in X.$$

A set of attribute implications is called a *theory*.

A set of attributes M is called a *model* of theory $\mathcal{T}$ if every attribute implication in $\mathcal{T}$ is valid in M. The set of all models of $\mathcal{T}$ is denoted $\mathrm{Mod}(\mathcal{T})$, i.e.

$$\mathrm{Mod}(\mathcal{T}) = \{M \mid \forall L \Rightarrow R \in \mathcal{T} : \|L \Rightarrow R\|_M = 1\}.$$

For any theory $\mathcal{T}$, the set $\mathrm{Mod}(\mathcal{T})$ of its models is a closure system. The corresponding closure operator, $c_{\mathrm{Mod}(\mathcal{T})}$, is equal to the following operator $c_{\mathcal{T}}$. For $Z \subseteq Y$ and theory $\mathcal{T}$, put

1. $Z^{\mathcal{T}} = Z \cup \bigcup\{R \mid L \Rightarrow R \in T, L \subseteq Z\}$,
2. $Z^{\mathcal{T}_0} = Z$,
3. $Z^{\mathcal{T}_n} = (Z^{\mathcal{T}_{n-1}})^{\mathcal{T}}$.

Define operator $c_{\mathcal{T}} : 2^Y \to 2^Y$ by

$$c_{\mathcal{T}}(Z) = \bigcup_{n=0}^{\infty} Z^{\mathcal{T}_n}.$$

A theory $\mathcal{T}$ is called

- *complete* in $\langle X, Y, I\rangle$ if $\mathrm{Mod}(\mathcal{T}) = \mathrm{Int}^{\uparrow\downarrow}(X, Y, I)$;
- a *basis* of $\langle X, Y, I\rangle$ if no proper subset of $\mathcal{T}$ is complete in $\langle X, Y, I\rangle$.

A set $P \subseteq Y$ of attributes is called a *pseudo-intent* if it satisfies the following conditions:

(i) it is not an intent, i.e. $P^{\downarrow\uparrow} \neq P$;

(ii) for all smaller pseudo-intents $P_0 \subset P$, we have $P_0^{\downarrow\uparrow} \subset P$.

Theorem 1. *Let $\mathcal{P}$ be a set of all pseudo-intents of $\langle X, Y, I\rangle$. The set*

$$\{P \Rightarrow P^{\downarrow\uparrow} \mid P \in \mathcal{P}\}$$

is a basis of $\langle X, Y, I\rangle$. Additionally, it is a minimal basis in terms of the number of attribute implications.

The basis from Theorem 1 is called the *Duquenne-Guigues basis* (DG basis).

Let $\mathcal{P}$ be a set of all pseudo-intents of $\langle X, Y, I\rangle$. The union $\mathrm{Int}^{\uparrow\downarrow}(I) \cup \mathcal{P}$ is a closure system on Y.

The corresponding closure operator $\tilde{c}_{\mathcal{T}}$ is given as follows. For $Z \subseteq Y$ and theory $\mathcal{T}$, put

1. $Z^{\mathcal{T}} = Z \cup \bigcup\{R \mid L \Rightarrow R \in \mathcal{T}, L \subset Z\}$,

2. $Z^{\mathcal{T}_0} = Z$,

3. $Z^{\mathcal{T}_n} = (Z^{\mathcal{T}_{n-1}})^{\mathcal{T}}$.

Define operator $\tilde{c}_{\mathcal{T}} : 2^Y \to 2^Y$ by

$$\tilde{c}_{\mathcal{T}}(Z) = \bigcup_{n=0}^{\infty} Z^{\mathcal{T}_n}. \tag{4}$$

Note that the definition of $\tilde{c}_{\mathcal{T}}$ differs from the previously defined $c_{\mathcal{T}}$ only in the subsethood in item 1 – the operator $c_{\mathcal{T}}$ allows equality in this item while $\tilde{c}_{\mathcal{T}}$ does not. In what follows, we use the shortcut $Z^{\bullet}$ for $\tilde{c}_{\mathcal{T}}(Z)$.

The algorithm which follows the above definition is called the naïve algorithm. There are more sophisticated ways to compute closures, like LinClosure [70] and Wild's closure [89].

To compute the closure system $\mathrm{Int}^{\uparrow\downarrow}(I) \cup \mathcal{P}$ using the above closure operator, the intents and pseudo-intents must be enumerated in an order $\leq$ which extends the subsethood; i.e.

$$C_1 \subseteq C_2 \text{ implies } C_1 \leq C_2 \qquad \text{for all } C_1, C_2 \in \mathrm{Int}^{\uparrow\downarrow}(I) \cup \mathcal{P}. \tag{5}$$

The lectic order satisfies this condition; that is why NextClosure [36] is most frequently used for this task.

1.2.3 Two-valued setting

For the explanation of the link between formal concept analysis and logical analysis of data, we need to recall the particular generalization of FCA, which is called two-valued FCA, three-way FCA [77, 78] or FCA with positive and negative attributes [79, 80]. In two-valued FCA, we assume a slightly different meaning of the input context. Specifically, we assume that the input context is two-valued. That means that the semantics of the relation I is as follows.

- $\langle x, y \rangle \in I$ means that the object x has the attribute y (as in the one-valued setting)
- $\langle x, y \rangle \notin I$ means that the object x does not have the attribute y (which is not necessary in the case for the one-valued setting).

The concept-forming operators in two-valued setting are defined as mappings ${}^{\vartriangle} : \mathbf{2}^X \to \mathbf{2}^Y \times \mathbf{2}^Y$ and ${}^{\triangledown} : \mathbf{2}^Y \times \mathbf{2}^Y \to \mathbf{2}^X$ given as

$$\begin{aligned} A^{\vartriangle} &= \langle A^{\uparrow}, A^{\cap} \rangle \quad \text{for } A \subseteq X, \\ \langle \underline{B}, \overline{B} \rangle^{\triangledown} &= \underline{B}^{\downarrow} \cap \overline{B}^{\cup} \quad \text{for } \underline{B}, \overline{B} \subseteq Y. \end{aligned} \tag{6}$$

The symbols ${}^{\cap}, {}^{\cup}$ in (6) denote the following operators:

$^{\cap}: \mathbf{2}^X \to \mathbf{2}^Y$ assigns to a set A of objects the set $A^{\cap}$ of all attributes which at least one object in A has.

$^{\cup}: \mathbf{2}^Y \to \mathbf{2}^X$ assigns to a set B of attributes the set $B^{\cup}$ of all object which have no attributes other than those in B.

Formally, for all $A \subseteq X, B \subseteq Y$, we have

$$\begin{aligned} A^{\cap} &= \{y \in Y \mid \exists x \in A : \langle x, y\rangle \in I\}, \\ B^{\cup} &= \{x \in X \mid \forall y \in Y : \langle x, y\rangle \in I \text{ implies } y \in B\}. \end{aligned}$$

We call triples $\langle A, \underline{B}, \overline{B}\rangle \in \mathbf{2}^X \times \mathbf{2}^Y \times \mathbf{2}^Y$ satisfying

$$A^{\vartriangle} = \langle \underline{B}, \overline{B}\rangle \quad \text{and} \quad \langle \underline{B}, \overline{B}\rangle^{\triangledown} = A \tag{7}$$

two-valued concepts. The set A in a concept $\langle A, \underline{B}, \overline{B}\rangle$ is then called the *extent* and the pair of sets $\langle \underline{B}, \overline{B}\rangle$ is called the *intent*.

We will utilize the following properties of the two-valued concept-forming operators.

Lemma 1. *For all $A \in \mathbf{2}^X, \underline{B}, \overline{B} \in \mathbf{2}^Y$, we have*

$$A^{\vartriangle\triangledown\vartriangle} = A^{\vartriangle} \qquad \textit{and} \qquad \langle \underline{B}, \overline{B}\rangle^{\triangledown\vartriangle\triangledown} = \langle \underline{B}, \overline{B}\rangle^{\triangledown}.$$

We denote the set of all two-valued concepts in $\langle X, Y, I\rangle$ by $\mathcal{B}(I)$. We denote by $\mathrm{Ext}(I)$ and $\mathrm{Int}(I)$ the set of all extents and the set of all intents in $\mathcal{B}(I)$, respectively.

On $\mathcal{B}(I)$ we define an order $\leq$ by

$$\begin{aligned} \langle A_1, \underline{B}_1, \overline{B}_1\rangle \leq \langle A_2, \underline{B}_2, \overline{B}_2\rangle \quad &\text{if} \quad A_1 \subseteq A_2, \\ &\text{equivalently, if } \underline{B}_1 \supseteq \underline{B}_2 \text{ and } \overline{B}_1 \subseteq \overline{B}_2. \end{aligned} \tag{8}$$

The set $\mathcal{B}(I)$ endowed with $\leq$ forms a complete lattice, called a *concept lattice*, in which infima and suprema are given by[1]

$$\bigwedge_{j \in J} \langle A_j, \underline{B}_j, \overline{B}_j\rangle = \langle \bigcap_{j \in J} A_j, \langle \bigcup_{j \in J} \underline{B}_j, \bigcap_{j \in J} \overline{B}_j\rangle^{\triangledown\vartriangle}\rangle, \tag{9}$$

$$\bigvee_{j \in J} \langle A_j, \underline{B}_j, \overline{B}_j\rangle = \langle (\bigcup_{j \in J} A_j)^{\vartriangle\triangledown}, \bigcap_{j \in J} \underline{B}_j, \bigcup_{j \in J} \overline{B}_j\rangle \tag{10}$$

respectively, for all $A_j \in \mathbf{2}^X, \underline{B}_j, \overline{B}_j \in \mathbf{2}^Y$ (J is an index set).

[1] In (9), we naturally unify triples of the form $\langle A, \underline{B}, \overline{B}\rangle$ with pairs of the form $\langle A, \langle \underline{B}, \overline{B}\rangle\rangle$.

Remark 1 (Reduction to one-valued setting)**.** *Two-valued setting of FCA is easy to reduce to one-valued setting. Specifically, one can create a one-valued formal context $\langle X, Y^*, I^*\rangle$ for a two-valued formal context $\langle X, Y, I\rangle$ as follows. We add a negative attribute $\hat{y}$ for each original attribute $y \in Y$, i.e.*

$$Y^* = Y \cup \hat{Y}$$

where

$$\hat{Y} = \{\hat{y} \mid y \in Y\}.$$

Furthermore, we extend I to the new attributes by putting

$$\langle x, \hat{y}\rangle \in I^* \quad \textit{iff} \quad \langle x, y\rangle \notin I \qquad \textit{for } x \in X, y \in Y.$$

The concept lattice $\mathcal{B}^{\uparrow\downarrow}(I^*)$ of the one-valued context $\langle X, Y^*, I^*\rangle$ is isomorphic to $\mathcal{B}(I)$ of the two-valued formal context $\langle X, Y, I\rangle$. Particularly,

$$\mathrm{Ext}^{\uparrow\downarrow}(I) = \mathrm{Ext}(I^*)$$

and the isomorphism $i : \mathcal{B}^{\uparrow\downarrow}(I^*) \to \mathcal{B}(I)$ is given by

$$i : \langle A, B\rangle \mapsto \langle A, B \cap Y, \{y \in Y \mid \hat{y} \notin B\}\rangle.$$

In words, having a one-valued formal concept $\langle A, B\rangle$, we obtain the corresponding two-valued formal concept $\langle A, \underline{B}, \overline{B}\rangle$ as follows:

- *the extent A remains the same;*
- *the first set $\underline{B}$ of the two-valued intent contains original (positive) attributes from the intent B;*
- *the second set $\overline{B}$ of the two-valued intent contains those attributes y, which do not have their negative attribute $\hat{y}$ in B.*

Example 1. *Figure 2 shows the one-valued context $\langle X, Y^*, I^*\rangle$ of the two-valued context $\langle X, Y, I\rangle$ from Fig. 1. The pair $\langle\{\mathrm{b}, \mathrm{c}, \mathrm{d}\}, \{3, \hat{4}\}\rangle$ is a formal concept of $\langle X, Y^*, I^*\rangle$. Its corresponding two-valued concept is*

$$\langle\{\mathrm{b}, \mathrm{c}, \mathrm{d}\}, \{3\}, \{1, 2, 3, 5\}\rangle$$

since

$$\{3, \hat{4}\} \cap Y = \{3\} \quad \textit{and} \quad Y \setminus \{4\} = \{1, 2, 3, 5\}.$$

	1	$\hat{1}$	2	$\hat{2}$	3	$\hat{3}$	4	$\hat{4}$	5	$\hat{5}$
a	1	0	0	1	1	0	1	0	1	0
b	1	0	1	0	1	0	0	1	1	0
c	1	0	1	0	1	0	0	1	0	1
d	0	1	0	1	1	0	0	1	1	0

Figure 2: One-valued context $\langle X, Y^*, I^*\rangle$ corresponding to two-valued context $\langle X, Y, I\rangle$ from Fig. 1.

1.2.4 Intents as intervals

For what follows in the Section 4, it is important to observe that for intents $\langle \underline{B}, \overline{B}\rangle$ in $\mathcal{B}(I)$, we have that $\underline{B} \subseteq \overline{B}$. The only exception is the intent of the bottom concept when its extent is empty. Thus, we can consider the intents to be intervals $[\underline{B}, \overline{B}]$ in $\mathbf{2}^Y$. The exceptional intent then corresponds to an empty interval. Additionally, if we have a pair $\langle \underline{B}, \overline{B}\rangle \in \mathbf{2}^Y \times \mathbf{2}^Y$ such that $\underline{B} \not\subseteq \overline{B}$, we clearly have that $\langle \underline{B}, \overline{B}\rangle^\triangledown = \emptyset$.

In the rest of the paper, we consider the intents of $\mathcal{B}(I)$ to be intervals in $\mathbf{2}^Y$. We denote the set of all intervals on Y by $\mathcal{I}^Y$.

On $\mathcal{I}^Y$, we consider two operations:

- intersection of intervals

$$\bigsqcap_{j\in J}[\underline{B}_j, \overline{B}_j] = [\bigcup_{j\in J}\underline{B}_j, \bigcap_{j\in J}\overline{B}_j],$$

- consensus of intervals

$$\bigsqcup_{j\in J}[\underline{B}_j, \overline{B}_j] = [\bigcap_{j\in J}\underline{B}_j, \bigcup_{j\in J}\overline{B}_j],$$

for all $[\underline{B}_j, \overline{B}_j] \in \mathcal{I}^Y$ (J is an index set).

Remark 2 (Notation)**.** *To simplify notation, we denote intervals by boldface italic capital letters instead of bracketed pairs (e.g.* $\boldsymbol{B}$ *instead of* $[\underline{B}, \overline{B}]$*).*

Using the above remarks, we can describe infima (9) and suprema (10) in a two-valued concept lattice as

$$\bigwedge_{j\in J}\langle A_j, \boldsymbol{B}_j\rangle = \langle \bigcap_{j\in J} A_j, (\bigsqcap_{j\in J}\boldsymbol{B}_j)^{\triangledown\vartriangle}\rangle,$$
$$\bigvee_{j\in J}\langle A_j, \boldsymbol{B}_j\rangle = \langle (\bigcup_{j\in J} A_j)^{\vartriangle\triangledown}, \bigsqcup_{j\in J}\boldsymbol{B}_j\rangle$$

respectively, for all $\langle A_j, \boldsymbol{B}_j\rangle \in \mathcal{B}(I)$ (J is an index set).

1.3 Selected algorithms used in FCA

In this section we describe two algorithms which are necessary for the rest of the paper.

1.3.1 Close-by-One

Close-by-One (CbO) is efficient algorithm for computing closure systems designed by Sergei Kuznetsov [62]. Since the set of all intents $\mathrm{Int}^{\uparrow\downarrow}(I)$ of formal context $\langle X, Y, I \rangle$ is closure system, we can use CbO for enumerating all intents. And what is more we can use CbO for computing DG basis of $\langle X, Y, I \rangle$ since intents and pseudo-intents also forms closure system.

Many algorithms for computing closure systems exist [67]. Among the most efficient algorithms are variants of CbO, namely Outrata & Vychodil's FCbO [73] and Andrews's In-Close family of algorithms [5, 6, 7, 8, 9].

In this section we briefly describe the CbO algorithm.

We assume a closure operator c on set $Y = \{1, 2, \ldots, n\}$. Whenever we write about lower attributes or higher attributes, we refer to the natural ordering of the numbers in Y.

The algorithm is given by a recursive procedure `CbOStep`, which accepts two arguments:

- B – the set of attributes, from which new sets will be generated.
- y – the auxiliary argument to remember the highest attribute in B.

One can see the pseudocode of CbO in Algorithm 1. The check of the condition $D_i = B_i$, where $D_i = D \cap \{1, \ldots, i-1\}$ is called a *canonicity test* (line 4).

Algorithm 1: Close-by-One

```
  def CbOStep(B, y):
      input : B – closed set
              y – last added attribute

1     print(B)
2     for i ∈ {y + 1, ..., n} \ B do
3         D ← c(B ∪ {i})
4         if D_y = B_y then
5             CbOStep(D, i)

  CbOStep(∅↓↑, 0)
```

We have some remarks for the algorithm:

- The argument B is a closed set, therefore, the procedure `CbOStep` can print it directly without testing (line 1).
- In the loop, we skip elements already present in B (line 2).
- The recursive invocation is made only if the the new closed set D passes the canonicity test (lines 3,4).

1.3.2 LinClosure

LinClosure (Algorithm 2) [13, 70] accepts a set B of attributes for which it computes the $\mathcal{T}$-closure $\tilde{c}_{\mathcal{T}}(B)$. The theory T is considered to be a global variable. It starts with a set D containing all elements of B (line 1). If there is an attribute implication in $\mathcal{T}$ with empty left side, the D is unified with its right side (lines 2,3). LinClosure associates a counter $count[L \Rightarrow R]$ with each $L \Rightarrow R \in \mathcal{T}$ initializing it with the size $|L|$ of its left side (lines 4,5). Also, each attribute $y \in Y$ is linked to a list of the attribute implications that have y in their left sides (lines 6,7).[2] Then, the set Z of attributes to be processed is initialized as a copy of the set D (line 8). While there are attributes in Z, the algorithm chooses one of them (min in the pseudocode, line 10), removes it from Z (line 11) and decrements counters of all attribute implication linked to it (lines 12,13). If the counter of any attribute implication $L \Rightarrow R$ is decreased to 0, new attributes from R are added to D and to Z.

We are going to use the algorithm LinClosure in CbO. CbO drops the resulting closed set if it fails the canonicity test (Algorithm 1, lines 4,5). Therefore, we can introduce a feature which stops the computation whenever an attribute which would cause the fail is added into the set. To do that, we add a new input argument, y, having the same role as in CbO; i.e. the last attribute added into the set. Then, whenever new attributes are added to the set, we check whether any of them is lower than y. If so, we stop the procedure and return information that the canonicity test would fail (lines 16–18).[3]

1.4 Logical analysis of data

Logical analysis of data (LAD) [1, 26, 29] is a method of binary data analysis, developed at Rutgers University by Peter L. Hammer and his colleagues. It produces accurate, reproducible, and robust classification models with high explanatory power; the accuracy of LAD models compares favorably with that of other machine learning and statistical models [20, 4, 2]. LAD has been

[2] This needs to be done just once and it is usually done outside the LinClosure procedure.
[3] This feature is also utilized in [12].

Algorithm 2: LinClosure with an early stop

```
def LinClosureES(B, y):
    input : B – set of attributes
            y – last attribute added to B
    D ← B
    if ∃∅ ⇒ R ∈ 𝒯 for some R then
        D ← D ∪ R
    for all L ⇒ R ∈ 𝒯 do
        count[L ⇒ R] ← |L|
        for all a ∈ L do
            add L ⇒ R to list[a]
    Z ← D
    while Z ≠ ∅ do
        m ← min(Z)
        Z ← Z \ {m}
        for all L ⇒ R ∈ list[m] do
            count[L ⇒ R] ← count[L ⇒ R] − 1
            if count[L ⇒ R] = 0 then
                add ← R \ D
                if min(add) < y then
                    return fail
                else
                    D ← D ∪ add
                    Z ← Z ∪ add
    return D
```

		1	2	3	4	5
	a	1	0	1	1	1
	b	1	1	1	0	1
Ω^+	c	1	1	1	0	0
	d	0	0	1	0	1
	e	1	0	1	0	0
Ω^-	f	1	0	0	0	0
	g	0	0	1	0	0

Figure 3: A binary dataset $\langle \Omega^+, \Omega^- \rangle$.

applied to numerous disciplines, e.g. credit risk ratings [41, 42], show rate prediction in the airline industry [33], fault detection and diagnosis for condition based maintenance [91], screening for growth hormone deficiency [69], labor productivity estimation [40], and probabilistic discrete choice models [21], to name just a few. Recent achievements of LAD are summarized by Miguel Lejeune *et al.* in their review paper [68].

Original versions of LAD were designed for the analysis of binary data. Binary data appear in the form of vertices of n-dimensional unit cube $\mathbf{2}^n$ (i.e. n-dimensional binary vectors) called *observations*. Components of the observations are called *attributes* (or *features*, or sometimes *variables*). Each observation is labeled as positive or negative. The set of all positive observations is denoted by Ω^+ and the set of all negative observations is denoted by Ω^- (see Fig. 3 for an example).

Remark 3. *As observations are n-dimensional binary vectors, we unify them with (characteristic vectors of) sets in the universe $Y = \{1, 2, \ldots, n\}$.*

For an interval $\boldsymbol{B}$ in the n-dimensional unit cube, we define a *coverage* $\operatorname{Cov}(\boldsymbol{B})$ as a set of observations contained in $\boldsymbol{B}$; that is

$$\operatorname{Cov}(\boldsymbol{B}) = \boldsymbol{B} \cap \Omega.$$

Analogously, we define positive and negative coverage respectively as

$$\operatorname{Cov}^+(\boldsymbol{B}) = \boldsymbol{B} \cap \Omega^+ \qquad \text{and} \qquad \operatorname{Cov}^-(\boldsymbol{B}) = \boldsymbol{B} \cap \Omega^-.$$

A basic notion in LAD is that of a pattern. An interval $\boldsymbol{B}$ in the n-dimensional unit cube is called a *positive pattern* if

$$\boldsymbol{B}^{\nabla_+} \neq \emptyset \quad \text{and} \quad \boldsymbol{B}^{\nabla_-} = \emptyset.^4$$

[4] For understanding this notation, please see Idea 1 in Section 4.

In words, the interval contains at least one positive observation and no negative observations. A *negative pattern* has an analogous definition.

In the rest of the paper, when we write just 'pattern' we mean either a positive or negative pattern.

A pattern $\boldsymbol{P}$ is *prime* if there is no pattern $\boldsymbol{P}'$ such that $\boldsymbol{P} \subset \boldsymbol{P}'$, i.e. if any enlargement of $\boldsymbol{P}$ results in an interval which is not a pattern. A pattern $\boldsymbol{P}$ is *strong* if there is no pattern $\boldsymbol{P}'$ such that $\boldsymbol{P}^{\triangledown} \subset \boldsymbol{P}'^{\triangledown}$.

Let T be a subset of observations. An *interval spanned by* T, denoted by $\mathrm{Span}(T)$, is the smallest interval containing all observations in T. That is,

$$\mathrm{Span}(T) = \left[\bigcap T, \bigcup T\right]. \tag{11}$$

If a pattern is spanned by a subset T, we call it a *spanned pattern*. The set of all spanned patterns in Ω is denoted by $\mathrm{SPAN}(\Omega)$, the set of all positive patterns and the set of all negative patterns are denoted by $\mathrm{SPAN}^+(\Omega)$ and $\mathrm{SPAN}^-(\Omega)$, respectively.

2 LinCbO: fast algorithm for computation of the Duquenne-Guigues basis

In this section, we describe the LinCbO algorithm. Its foundation is CbO (Algorithm 1) with LinClosure (Algorithm 2). When considering systems of attribute implications, pseudo-intents play an important role, since they derive the minimal basis, called the Duquenne-Guigues basis or canonical basis [39]. The pseudo-intents, together with the intents of formal concepts, form a closure system. Enumerating all pseudo-intents (together with intents) is more challenging as it requires a particular restriction of the order of the computation and the results on complexity are all but promising [64]. There are basically two main approaches for this task: NextClosure by Ganter [35, 36], and the incremental approach by Obiedkov and Duquenne [72].

We show that in our approach, LinClosure is able to reuse attribute counters.[5] from previous computations. This makes it work very fast, as our experiments show.

We explain changes in the CbO algorithm: a change of sweep order makes the algorithms work, and the rest of the changes improve efficiency of the algorithms.

[5]LinClosure uses so-called attribute counters to avoid set comparisons and reach a linear time complexity. We recall this in Section 1.3.2.

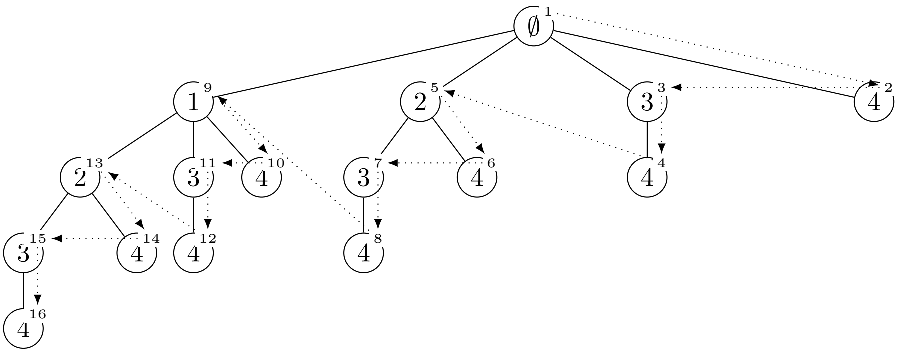

Figure 4: Tree of all subsets of $\{1, 2, 3, 4\}$. Each node represents a unique set containing all elements in the path from the node to the root. The dotted arrows and small numbers represent the sweep performed by the CbO algorithm with right depth-first sweep.

2.1 Sweep order

In the previous section, we presented CbO as the left first sweep through the tree of all subsets. This is how it is usually described. In ordinary settings, there is no need to follow a particular order of sweep. However, our purpose is to compute intents and pseudo-intents using the closure operator $\tilde{c}_{\mathcal{T}}$ (4). For this, we need to utilize an order which extends the subsethood, i.e. (5). The right depth-first sweep through the tree of all subsets satisfies this condition (see Fig. 4). Observe that with the right depth-first sweep, we obtain exactly the lectic order, i.e. the same order in which NextClosure explores the search space.

2.2 NextClosure's improvements

The following improvements were introduced to NextClosure [12] and the incremental approach [72] for computation of pseudo-intents. We incorporated them to the CbO algorithm.

After the algorithm computes $B^{\bullet}$, the implication $B^{\bullet} \rightarrow B^{\downarrow\uparrow}$ is added to $\mathcal{T}$, provided $B^{\bullet}$ is a pseudo-intent, i.e. $B^{\bullet} \neq B^{\downarrow\uparrow}$.

Note that there exists the smallest $\tilde{c}_{\mathcal{T}}$-closed set larger than $B^{\bullet}$ and it is the intent $B^{\bullet\downarrow\uparrow}$ ($= B^{\downarrow\uparrow}$). Consider the following two cases:

($o1$) This intent satisfies the canonicity test, i.e. $B_y^{\downarrow\uparrow} = B_y^{\bullet}$, where y is the last added attribute to A. Then we can jump to this intent.

($o2$) This intent does not satisfy the canonicity test. Thus, we can leave the present subtree.

Now, let us describe the first version of LinCbO (Algorithm 3), which includes the above discussed improvements.

The procedure `LinCbO1Step` works with the following global variables: an initially empty theory $\mathcal{T}$, and an initially empty list of attribute implication for each attribute. `LinCbO1Step` accepts two arguments: a set B of attributes and the last attribute y added to B. The set B is not generally closed (which was the case in Algorithm 1).

The procedure first applies LinClosure with an early stop (Algorithm 2) to compute $B^\bullet$ (line 1).

If $B^\bullet$ fails the canonicity test (recall that the canonicity test is incorporated in LinClosure with an early stop), the procedure stops (lines 2,3). Then, the procedure computes $B^{\bullet\downarrow\uparrow}$ to check whether $B^\bullet$ is an intent or pseudo-intent (line 4). If it is a pseudo-intent, a new attribute implication $B^\bullet \Rightarrow B^{\bullet\downarrow\uparrow}$ is added to the initially empty theory $\mathcal{T}$ (line 5). For each attribute in $B^\bullet$, we update its list by adding the new attribute implication (lines 6 and 7).

Now, as we computed the intent $B^{\bullet\downarrow\uparrow}$, we can apply (o1) or (o2) based on the result of the canonicity test $B_y^{\bullet\downarrow\uparrow} = B_y^\bullet$ (line 8) – either we call `LinCbO1Step` for $B^{\bullet\downarrow\uparrow}$ (line 9) or end the procedure. If $B^\bullet$ is an intent, we recursively call `LinCbO1Step` for all sets $B^\bullet \cup \{i\}$ where i is higher than the last added attribute y and is not already present in $B^\bullet$. To have lectic order, we make the recursive calls in the descending order of is.

The procedure `LinCbO1Step` is initially called with empty set of attributes and zero representing an invalid last added attribute.

2.3 LinClosure with reused counters

Consider theory $\mathcal{T}'$ and theory $\mathcal{T}$ which emerges by adding new attribute implications to $\mathcal{T}'$, i.e. $\mathcal{T}' \subseteq \mathcal{T}$. When we compute $\mathcal{T}'$-closure B', we can store values of the attribute counters at the end of the LinClosure procedure. Later, when we compute $\mathcal{T}$-closure of a superset B of B', we can initialize the attribute counters of implications from $\mathcal{T}'$ to the stored values instead of the antecedent sizes. Attribute counters for new implications, i.e. those in $\mathcal{T}' \setminus \mathcal{T}$, are initialized the usual way. Then, we handle only the new attributes, that is those in $B \setminus B'$.

We can improve the LinClosure accordingly (Algorithm 4). We describe only the differences from LinClosure with an early stop (Algorithm 2). It accepts two additional arguments: Z – the set of new attributes, i.e, those which were not in the $\mathcal{T}$-closed subset from which we reuse the counters; and $prevCount$ – the previous counters to be reused. We copy the previous counters (line 4) and add new attribute implications (lines 5,6).

Algorithm 3: LinCbO1 (CbO for the Duquenne-Guigues basis, first version)

$\mathcal{T} \leftarrow \emptyset$
$list[i] \leftarrow \emptyset$ **for each** $i \in Y$

def `LinCbO1Step`(B, y):
  input : B – set of attributes
    y – last attribute added to B

1 $B^\bullet \leftarrow$ `LinClosureES`(B, y)
2 **if** $B^\bullet$ *is fail* **then**
3   return
4 **if** $B^\bullet \neq B^{\bullet\downarrow\uparrow}$ **then**
5   $\mathcal{T} \leftarrow \mathcal{T} \cup \{B^\bullet \Rightarrow B^{\bullet\downarrow\uparrow}\}$
6   **for** $i \in B^\bullet$ **do**
7     $list[i] \leftarrow list[i] \cup \{B^\bullet \Rightarrow B^{\bullet\downarrow\uparrow}\}$
8   **if** $B_y^{\bullet\downarrow\uparrow} = B_y^\bullet$ **then**
9   `LinCbO1Step`$(B^{\bullet\downarrow\uparrow}, y)$
10 **else**
11   **for** i **from** n **down to** $y + 1$, $i \notin B^\bullet$ **do**
12   `LinCbO1Step`$(B^\bullet \cup \{i\}, i)$

`LinCbO1Step`$(\emptyset, 0)$

Note, that in CbO we always make the recursive invocations for supersets of the current set (see Algorithm 3, lines 9 and 12). Therefore, we can easily utilize the LinClosure with reused counters in LinCbO (Algorithm 5). The only difference from the first version (Algorithm 3) is that the procedure `LinCbOStep` accepts two additional arguments, which are passed to procedure `LinClosureRC` (line 1). The two arguments are: the set of new attributes and the previous attribute counters (both initially empty). Recall that the attribute counters are modified by LinClosure. The corresponding arguments are also passed to the recursive invocations of `LinCbOStep` (lines 9 and 12).

2.4 Experimental comparison

We compare LinCbO with other algorithms, namely:

- NextClosure with naïve closure (NC1), LinClosure (NC2), and Wild's closure (NC3).

Algorithm 4: LinClosure with reused counters

```
def LinClosureRC(B, y, Z, prevCount):
    input : B – set of attributes to be closed
            y – last attribute added to B
            Z – set of new attributes
            prevCount – previous attribute counters from
    computation B \ Z

    D ← B
    if ∃∅ ⇒ R ∈ T then
        D ← D ∪ R
    count ← copy of prevCount
    for L ⇒ R ∈ T not counted in prevCount do
        count[L ⇒ R] ← |L \ B|
    while Z ≠ ∅ do
        m ← min(Z)
        Z ← Z \ {m}
        for L ⇒ R ∈ list[m] do
            count[L ⇒ R] ← count[L ⇒ R] − 1
            if count[L ⇒ R] = 0 then
                add ← R \ D
                if min(add) < y then
                    return fail
                D ← D ∪ add
                Z ← Z ∪ add
    return ⟨D, count⟩
```

- NextClosure$^+$, which is NextClosure with the improvements described in Section 2.2, with the same closures (NC$^+$1, NC$^+$2, NC$^+$3)[6];
- attribute incremental approach [72].

To achieve maximal fairness, we implemented LinCbO into the framework made by Bazhanov & Obiedkov [12][7]. It contains implementations of all the listed algorithms. In Section 2.4.1, we also use the same datasets as used by Bazhanov and Obiedkov [12].

All experiments have been performed on a computer with 64 GB RAM, two

[6] NextClosure and NextClosure$^+$ are called Ganter and Ganter$^+$ in [12].
[7] Available at `https://github.com/yazevnul/fcai`

Algorithm 5: LinCbO (CbO for the Duquenne-Guigues basis, final version)

$\mathcal{T} \leftarrow \emptyset$
$list[i] \leftarrow 0$ **for each** $y \in Y$

def `LinCbOStep`(B, y, Z, $prevCount$):
  input : B – set of attributes
    y – last attribute added to B
    Z – set of new attributes
    $prevCount$ – attribute counters

  $\langle B^\bullet, count\rangle \leftarrow$ `LinClosureRC`$(B, y, Z, prevCount)$
  if $B^\bullet$ *is fail* **then**
    return
  if $B^\bullet \neq B^{\bullet\downarrow\uparrow}$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{B^\bullet \Rightarrow B^{\bullet\downarrow\uparrow}\}$
    for $i \in B^\bullet$ **do**
      $list[i] \leftarrow list[i] \cup \{B^\bullet \Rightarrow B^{\bullet\downarrow\uparrow}\}$
    if $B_y^{\bullet\downarrow\uparrow} = B_y^\bullet$ **then**
      `LinCbOStep`$(B^{\bullet\downarrow\uparrow}, y, B^{\bullet\downarrow\uparrow} \setminus B^\bullet,\ count)$
  else
    for i **from** n **down to** $y+1$, $i \notin B^\bullet$ **do**
      `LinCbOStep`$(B^\bullet \cup \{i\}, i, \{i\}, count)$

`LinCbOStep`$(\emptyset, 0, \emptyset, \emptyset)$

Intel Xeon CPU E5-2680 v2 (at 2.80 GHz), Debian Linux 10, and GNU GCC 8.3.0. All measurements have been taken ten times and the mean value is presented.

2.4.1 Batch 1: datasets used in [12]

Bazhanov and Obiedkov [12] use artificial datasets and datasets from UC Irvine Machine Learning Repository [32].

The artificial datasets are named as $|X|$`x`$|Y|$-d, where d is the number of attributes of each object; i.e. $|x^\uparrow| = d$ for each $x \in X$. The attributes are assigned to objects randomly, with exception `18x18-17`, where each object misses a different attribute (more exactly, the incidence relation is the inequality).

The datasets from UC Irvine Machine Learning Repository are:

Table 1: Runtimes in seconds of algorithms generating Duquenne-Guigues basis in batch 1.

Dataset	AttInc	NC1	NC2	NC3	$NC^{+}1$	$NC^{+}2$	$NC^{+}3$	LinCbO
`100x30-4`	0.008	0.007	0.007	0.010	0.004	0.003	0.005	**0.002**
`100x50-4`	0.028	0.037	0.024	0.050	0.013	0.008	0.016	**0.005**
`10x100-25`	0.015	0.015	0.023	0.033	0.007	0.010	0.014	**0.004**
`10x100-50`	0.037	0.052	0.087	0.112	0.038	0.063	0.081	**0.015**
`18x18-17`	0.337	0.096	0.143	0.134	0.111	0.157	0.151	**0.148**
`20x100-25`	0.099	0.281	0.165	0.484	0.094	0.061	0.172	**0.026**
`20x100-50`	0.940	5.457	3.047	8.898	3.809	2.310	6.481	**0.675**
`50x100-5`	0.454	0.778	0.253	1.064	0.126	0.047	0.164	**0.029**
`900x100-4`	2.061	3.315	0.910	3.936	1.150	0.317	1.333	**0.172**
`Breast-cancer`	0.121	0.295	0.236	0.325	0.231	0.184	0.251	**0.055**
`Breast-w`	2.856	4.674	3.128	9.610	2.526	1.670	5.155	**0.516**
`dbdata0`	0.109	0.254	0.312	0.430	0.158	0.208	0.263	**0.049**
`flare`	0.622	1.006	1.865	1.813	0.920	1.661	1.624	**0.265**
`Post-operative`	0.014	0.015	0.023	0.021	0.013	0.018	0.018	**0.009**
`spect`	0.142	0.407	0.584	0.397	0.388	0.556	0.377	**0.097**
`vote`	0.054	0.062	0.078	0.068	0.059	0.075	0.064	**0.024**
`zoo`	0.004	0.003	0.005	0.005	**0.002**	0.004	0.004	**0.002**

`Breast-cancer`, `Breast-w`, `dbdata0`, `flare`, `Post-operative`, `spect`, `vote`, and `zoo`.

In batch 1, LinCbO computes the basis faster than the rest of algorithms; however in most cases the runtimes are very small and differences between them are negligible (see Table 1).

2.4.2 Batch 2: our collection of datasets

As the runtimes in batch 1 often differ only in a few milliseconds, we tested the algorithm on larger datasets. We used the following datasets from UC Irvine Machine Learning Repository [32]:

- `crx` – Credit Approval (37 rows containing a missing value were removed),
- `shuttle` – Shuttle Landing Control,
- `magic` – MAGIC Gamma Telescope,
- `bikesharing_(day|hour)` – Bike Sharing Dataset,
- `kegg` – KEGG Metabolic Reaction Network – Undirected.

We binarized the datasets using nominal (`nom`), ordinal (`ord`), and interordinal (`inter`) scaling, where each numerical feature was scaled to k attributes with $k-1$ equidistant cutpoints. Categorical features were scaled nominally to a number of attributes corresponding to the number of categories. After the binarization, we removed full columns. Properties of the resulting datasets are shown in Table 2. The naming convention used in Table 2 (and Table 3) is the following: (scaling)k(dataset). For example, `inter10shuttle` is the dataset 'Shuttle Landing Control' interordinally scaled to 10, using 9 equidistant cut-points.

Table 2: Properties of the datasets in batch 2

dataset	$\lvert X\rvert$	$\lvert Y\rvert$	$\lvert I\rvert$	# intents	# ps.intents
`inter10crx`	653	139	40,170	10,199,818	20,108
`inter10shuttle`	43,500	178	3,567,907	38,199,148	936
`inter3magic`	19,020	52	399,432	1,006,553	4181
`inter4magic`	19,020	72	589,638	24,826,749	21,058
`inter5bike_day`	731	93	24,650	3023,326	20,425
`inter5crx`	653	79	20,543	348,428	3427
`inter5shuttle`	43,500	88	1,609,510	333,783	346
`inter6shuttle`	43,500	106	2,002,790	381,636	566
`nom10bike_day`	731	100	9293	52,697	29,773
`nom10crx`	653	85	8774	51,078	6240
`nom10magic`	19,020	102	209,220	583,386	154,090
`nom10shuttle`	43,500	97	435,000	2931	810
`nom15magic`	19,020	152	209,220	1,149,717	397,224
`nom20magic`	19,020	202	209,220	1,376,212	654,028
`nom5bike_day`	731	65	9293	61,853	16,296
`nom5bike_hour`	17,379	90	238,292	1,868,205	320,679
`nom5crx`	653	55	8774	29,697	2162
`nom5keg`	65,554	144	1,834,566	13,262,627	42,992
`nom5shuttle`	43,500	52	435,000	1461	319
`ord10bike_day`	731	93	28,333	664,713	11,795
`ord10crx`	653	79	37,005	1,547,971	2906
`ord10shuttle`	43,500	88	1,849,216	97,357	279
`ord5bike_day`	731	58	14,929	81,277	5202
`ord5bike_hour`	17,379	83	457,578	2,174,964	99,691
`ord5crx`	653	49	19,440	139,752	973
`ord5magic`	19,020	42	535,090	821,796	1267
`ord5shuttle`	43,500	43	868,894	4068	119
`ord6magic`	19,020	52	662,177	2,745,877	2735

For this batch, we included LinCbO1 (Algorithm 3) to show how the reuse of attribute counters influences the performance.

For most datasets, LinCbO works faster than the other algorithms. For the remaining datasets, LinCbO is the second best after the attribute incremental approach (see Table 3). However, we encountered limits of the attribute incremental approach as it runs out of available memory in four cases (denoted by the symbol $*$ in Table 3).

2.4.3 Evaluation

Based on the experimental evaluation in Section 2.4, we conclude that LinCbO is the fastest algorithm for computation of the Duquenne-Guigues basis. In

Table 3: Runtimes in seconds of algorithms generating Duquenne-Guigues basis in batch 2. The symbol $*$ means that the run could not be completed due to insufficient memory

Dataset	AttInc	NC1	NC2	NC3	NC$^+$1	NC$^+$2	NC$^+$3	LinCbO	LinCbO1
inter10crx	**400.292**	2084.12	17,059.5	4256.41	2097.54	16,817.5	4193.46	508.551	23,842
inter10shuttle	$*$	18,038.1	21,268.1	20,211.9	17,664.5	21,035.4	20,171.9	**1585.9**	28,373.5
inter3magic	109.178	106.341	136.738	109.133	107.357	136.842	109.428	**26.156**	74.980
inter4magic	$*$	4029.95	9998.74	4241.51	4027.48	10,023	4239.26	**965.353**	9258.53
inter5bike_day	**72.952**	389.073	1409.69	680.789	383.537	1378.89	670.109	85.591	1589.58
inter5crx	5.863	16.357	56.977	25.08	16.257	56.669	24.995	**3.176**	75.205
inter5shuttle	207.323	137.211	144.747	144.125	137.596	145.491	144.957	**120.003**	143.4
inter6shuttle	253.166	164.355	181.19	177.138	164.924	182.664	178.474	**133.288**	181.967
nom10bike_day	**4.515**	42.074	33.725	71.745	31.505	24.710	52.249	7.099	26.318
nom10crx	1.227	3.105	5.409	7.776	2.828	4.792	6.859	**0.944**	6.939
nom10magic	486.926	1503.38	977.612	1547.33	1322.62	790.61	1246.06	**206.797**	821.269
nom10shuttle	1.455	1.14	1.19	1.234	1.102	1.134	1.166	**0.425**	0.53
nom15magic	3358.44	10,499.8	6442.54	14,838.1	8620.79	5060.17	11,277	**1509.86**	5363.77
nom20magic	7882.15	32,600.2	16,779.1	46,609.8	23,129.5	10,754.4	33,369.5	**4437.05**	17,424
nom5bike_day	2.58	13.064	11.32	17.572	10.855	9.383	14.517	**2.219**	9.251
nom5bike_hour	1893.33	8083.01	8412.02	8402.16	7248.4	7055.42	7163.17	**1410.11**	8098.72
nom5crx	0.406	0.623	1.054	1.061	0.592	0.983	0.988	**0.193**	1.110
nom5keg	$*$	7707.54	16,584.8	13,154.5	7564.71	16,590.3	13,184.1	**1936.7**	15,305
nom5shuttle	0.693	0.493	0.511	0.511	0.481	0.497	0.5	**0.309**	0.320
ord10bike_day	**21.884**	92.944	402.8	154.541	90.973	385.489	148.472	24.997	451
ord10crx	28.367	85.67	325.608	93.936	85.735	325.742	94.394	**11.653**	342.858
ord10shuttle	51.839	40.338	42.438	41.475	40.426	42.419	41.549	**34.293**	40.155
ord5bike_day	2.08	4.688	12.498	7.34	4.412	11.501	6.812	**0.936**	12.454
ord5bike_hour	1107.57	1749.29	5621.96	2304.73	1672.93	5173.36	2169.43	**321.147**	5694.64
ord5crx	1.468	2.7	6.696	3.071	2.701	6.680	3.062	**0.610**	6.957
ord5magic	99.92	93.845	108.648	94.280	93.930	108.733	94.437	**46.982**	71.721
ord5shuttle	1.676	1.382	1.408	1.410	1.380	1.403	1.404	**1.319**	1.417
ord6magic	345.392	335.947	447.37	337.462	336.4	447.353	338.321	**158.227**	277.617

some cases, it is outperformed by the attribute incremental approach. However, the attribute incremental approach seems to have enormous memory requirements as it run out of memory for several datasets.

Originally, we believed that CbO itself can make the computation faster. This motivation came from the paper by Outrata & Vychodil [73], where CbO is shown to be significantly faster than NextClosure when computing intents.

The main reason for the speed-up is the fact that CbO uses set intersection to efficiently obtain extents during the tree descent. This feature cannot be exploited for computation of the Duquenne-Guigues basis. The CbO itself rarely seems to have a significant effect on the runtime – this was the case for datasets `nom10shutle` and `nom5shutle`. Sometimes, it lead to worse performance, for example for datasets `inter10crx`, `inter10shuttle`, and `nom20magic`.

However, the introduction of the reuse of attribute counters significantly improves the runtime for most datasets (see Table 3).

2.5 Pruning in pseudointent computation

CbO received a few improvements in the last two decades, like parallel and distributed computation [55, 56], partial closures [5], or execution using the map-reduce framework [58, 54]. Arguably the most efficient improvement of CbO is a use of monotony property of closure operators to avoid some unnecessary computation of closures. This is utilized in FCbO [73], InClose-4 [8], InClose-5 [9], and LCM [86, 85, 87, 88, 49] (Section 3). We call these methods pruning techniques.

The operator $\tilde{c}_{\mathcal{T}}$ (4) is a closure operator; therefore it satisfies the monotony property, i.e. for any $B, D \subseteq 2^Y$ we have

$$B \subseteq D \text{ implies } \tilde{c}_{\mathcal{T}}(B) \subseteq \tilde{c}_{\mathcal{T}}(D). \tag{12}$$

Furthermore, for any two theories $\mathcal{T}$ and $\mathcal{S}$ with $\mathcal{T} \subseteq \mathcal{S}$, we have $\mathrm{Mod}(\mathcal{S}) \subseteq \mathrm{Mod}(\mathcal{T})$ and, consequently, for all $B \subseteq 2^Y$

$$\mathcal{T} \subseteq \mathcal{S} \text{ implies } \tilde{c}_{\mathcal{T}}(B) \subseteq \tilde{c}_{\mathcal{S}}(B). \tag{13}$$

Putting (12) and (13) together, we get that for any sets $B, D \subseteq 2^Y$ attributes and theories $\mathcal{T}$ and $\mathcal{S}$ we have

$$B \subseteq D \text{ and } \mathcal{T} \subseteq \mathcal{S} \text{ imply } \tilde{c}_{\mathcal{T}}(B) \subseteq \tilde{c}_{\mathcal{S}}(D). \tag{14}$$

From (14), we have that for any $i \in Y$:

$$B \subseteq D, \mathcal{T} \subseteq \mathcal{S} \text{ imply [if } i \in \tilde{c}_{\mathcal{T}}(B \cup \{y\}) \text{ then } i \in \tilde{c}_{\mathcal{S}}(D \cup \{y\})]. \tag{15}$$

Now, consider $B \cup \{y\}$ being a set to which y is added as the last attribute. Let i be an attribute with $i < y$ and $i \notin B$ and the theories $\mathcal{S}$ and $\mathcal{T}$ be

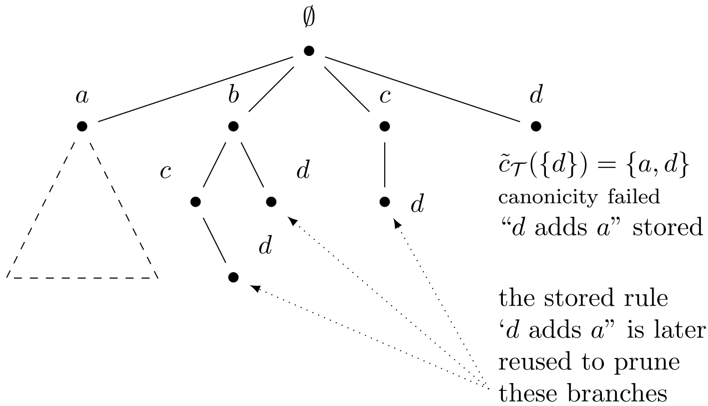

Figure 5: Idea of pruning

the partially computed Duquenne-Guigues basis in different times. Obviously, $i \in \tilde{c}_{\mathcal{T}}(B \cup \{y\})$ means that the closure of $\tilde{c}_{\mathcal{T}}(B \cup \{y\})$ fails the canonicity test. In words, (15) says that if the canonicity fails for $\tilde{c}_{\mathcal{T}}(B \cup \{y\})$ then it will also fail for $\tilde{c}_{\mathcal{S}}(D \cup \{y\})$.

We can store the information about failed canonicity test for $\tilde{c}_{\mathcal{T}}(B \cup \{y\})$ and use it later to avoid the computation of $\tilde{c}_{\mathcal{S}}(D \cup \{y\})$. This is what we call a pruning, as we effectively prune branches of the search tree.

Specifically, in our case, we store a rule of form "y adds i". This means that when we add the attribute y to the set B, the attribute i occurred in the closure $\tilde{c}_{\mathcal{T}}(B \cup \{y\})$ and caused the canonicity test to fail. We use the rule only in subtrees of B, as they contain only supersets of B.

Example 2. *In Figure 5 we illustrate a case, when $B = \emptyset$, $y = d$ and $i = a$. In the right-most branch, we observe that the canonicity test fails for $\emptyset \cup \{d\}$, because $a < d$ occurs in the closure $\tilde{c}_{\mathcal{T}}(\emptyset \cup \{d\})$. We store the rule "d adds a" and use it in subtrees of $\emptyset$ whenever we add d to a set. This enables us to avoid computation of $\tilde{c}_{\mathcal{T}}(\{c, d\})$, $\tilde{c}_{\mathcal{T}}(\{b, d\})$, and $\tilde{c}_{\mathcal{T}}(\{b, c, d\})$.*

2.5.1 Utilization of the pruning in LinCbO

We use a global array (*rules*) to store the rules for pruning. A rule "y adds i" is stored as $rules[y] = i$. Absence of such rule is represented by $rules[y] = 0$. Note that it means that a new rule can overwrite old rule if it has the attribute on the left side.

We need to modify LinClosure (recall that the canonicity test is incorporated in LinClosure) to provide us information for pruning. The modified LinClosure returns a triplet $\langle B^\bullet, fail, count\rangle$ where:

- $B^\bullet$ is the closed set if it passes the canonicity test.
- $fail$ the least attribute which violates the prefix if the canonicity test failed; otherwise it is 0.
- *count* are values of attribute counters.

In Algorithm 4, we only need to accordingly modify lines 15 and 18.

We also modify LinCbO as follows (refer to Algorithm 6):

(p0) Whenever the canonicity test fails `LinCboStep` returns the attribute $fail$, which LinClosure detected to violate the prefix (line 4). If an invocation of `LinCboStep` returns non-zero value $fail$, it stores the rule "y adds $fail$" in the array *rules* (lines 14-16).

(p1) At the beginning of `LinCboStep`, i.e. when descending to a subtree, all rules having the last added attribute (argument y) on the right side are removed from the stored rules. In the pseudocode, this is realized by a subroutine called `RemoveRulesByRightSide` (line 1).

(p2) At the end of `LinCboStep`, i.e. when backtracking from the current subtree, all rules from this call are removed. In the pseudocode, this is realized by a subroutine called `RemoveAllRulesAddedThisCall` (line 17).

(p3) Before computing a closure $\tilde{c}_{\mathcal{T}}(B^\bullet \cup \{i\})$ in a subtree of B, we check the stored rules to find whether adding i does not add an attribute which causes the canonicity test to fail (line 13).

We use two versions of the pruning:

`(lcm)`: does exactly what is described above. Notice that in (p3) it needs only to check existence of a rule with i on the left side; it needs not to check whether the attribute on its right side is in B (the part $rules[i] \in B^\bullet$ of the condition in line 13 of Algorithm 6 can be skipped).

`(lcmx)`: does what is described above but skips the step (p1) (line 1 of Algorithm 6 is skipped).

Remark 4. *Due to the early stop utilized in LinClosure, the information for pruning is not as complete as in the case for intents. We do not actually obtain* $\tilde{c}_{\mathcal{T}}(B^\bullet \cup \{y\})$ *used in* (15) *when the canonicity is violated. Instead we obtain an intermediate set. Still it is usable to form the pruning rules as at least one attribute causing the canonicity test to fail is present in the set.*

Algorithm 6: LinCbO, final version with pruning)

```
𝒯 ← ∅
list[i] ← 0 for each  y ∈ Y

def LinCbOStep(B, y, Z, prevCount):
    input  :  B – set of attributes
              y – last attribute added to B
              Z – set of new attributes
              prevCount – attribute counters

    RemoveRulesByRightSide(i)
    ⟨B•, fail, count⟩ ← LinClosureRC(B, y, Z, prevCount)
    if fail > 0 then
        return fail

    if B• ≠ B•↓↑ then
        𝒯 ← 𝒯 ∪ {B• ⇒ B•↓↑}
        for i ∈ B• do
            list[i] ← list[i] ∪ {B• ⇒ B•↓↑}
        if B_y•↓↑ = D_y then
            LinCbOStep(B•↓↑, y, B•↓↑ \ B•, count)
    else
        for i from n down to y + 1, i ∉ B• do
            if rules[i] = 0 or rules[i] ∈ B• then
                fail ←LinCbOStep(B• ∪ {i}, i, {i}, count)
                if fail > 0 then
                    rules[i] ← fail

    RemoveAllRulesStoredThisCall()
    return 0

LinCbOStep(∅, 0, ∅, ∅)
```

2.5.2 Experimental comparison

We experimentally compare three version LinCbO: without pruning and with the two pruning techniques described above. Additionally, we compare them with algorithms available in the framework made by Bazhanov & Obiedkov [12], namely Ganter, Ganter$^+$ – each with naïve closure, LinClosure [70], and Wild's closure [89] — and the attribute incremental approach.

All experiments have been performed on the same computer as in Section 2.4 with the datasets from batch 2 (Section 2.4.2).

Table 4: Runtimes in seconds of algorithms generating Duquenne-Guigues basis

Dataset	`LinCBO`	`LinCBO(lcm)`	`LinCBO(lcmx)`	best of the rest	
`inter10crx`	508.551	223.38	**199.115**	400.292	`AttInc`
`inter10shuttle`	15,852.9	14,967.7	**14,825.4**	17,664.5	`Ganter`$^+$
`inter3magic`	26.156	24.289	**24.192**	106.341	`Ganter`
`inter4magic`	965.353	**771.084**	835.315	4027.48	`Ganter`
`inter5bike_day`	85.591	44.012	**40.349**	72.952	`AttInc`
`inter5crx`	3.176	1.855	**1.802**	5.863	`AttInc`
`inter5shuttle`	120.003	**112.034**	112.638	137.211	`Ganter`
`inter6shuttle`	133.288	**126.91**	126.946	164.355	`Ganter`
`nom10bike_day`	7.099	1.682	**1.545**	4.515	`AttInc`
`nom10crx`	0.944	0.332	**0.328**	1.227	`AttInc`
`nom10magic`	206.797	96.377	**96.662**	486.926	`AttInc`
`nom10shuttle`	0.425	**0.382**	0.396	1.102	`Ganter`$^+$
`nom15magic`	1509.86	557.051	**544.459**	3358.44	`AttInc`
`nom20magic`	4437.05	1211.66	**1210.46**	7882.15	`AttInc`
`nom5bike_day`	2.219	0.833	**0.804**	2.580	`AttInc`
`nom5bike_hour`	1410.11	**476.592**	481.241	1893.33	`AttInc`
`nom5crx`	0.193	0.114	**0.106**	0.406	`AttInc`
`nom5keg`	1936.7	**1116.51**	1139.87	7564.710	`Ganter`$^+$
`nom5shuttle`	0.309	0.297	**0.292**	0.481	`Ganter`$^+$
`ord10bike_day`	24.997	15.947	**15.108**	21.884	`AttInc`
`ord10crx`	11.653	10.5153	**10.147**	28.367	`AttInc`
`ord10shuttle`	**34.293**	36.2858	36.2079	40.338	`Ganter`
`ord5bike_day`	0.936	0.713	**0.669**	2.080	`AttInc`
`ord5bike_hour`	321.147	273.862	**258.072**	1107.570	`AttInc`
`ord5crx`	0.610	0.559	**0.551**	1.468	`AttInc`
`ord5magic`	**46.982**	48.429	48.4259	93.845	`Ganter`
`ord5shuttle`	**1.319**	1.345	1.349	1.380	`Ganter`$^+$
`ord6magic`	**158.227**	158.466	162.65	335.947	`Ganter`

We made the following observations from our experimental evaluation (Table 4).

Comparison of LinCbO with and without pruning

The pruning techniques seem to have different effect for various types of formal contexts:

- For interordinally scaled data, LinCbO with pruning performed better than without pruning. However, the improvement is significant only for the `crx` datasets (`inter10crx` and `inter5crx`). For other dasasets, the improvement seems insignificant.
- For nominally scaled data, LinCbO with pruning performed significantly better with exception of `shuttle` dataset (`nom5shuttle` and `nom10shuttle`).
- For ordinally scaled data, LinCbO without pruning performed slightly better than with pruning – namely, for the `magic` and `shuttle` datasets (`ord5magic`, `ord6magic`, `ord10shuttle`, and `ord5shuttle`). LinCbO with pruning performed better in the rest. With exception for `ord10crx`, the improvement was significant.

The speed-up factor $\frac{\text{runtime of LinCbO with pruning}}{\text{runtime of LinCbO without pruning}} \cdot 100\%$ of the pruning methods is shown in Table 5.

Comparison of the two variants of pruning in LinCbO

The `(lcmx)` does not remove pruning rules in (p1) and enables them to be used until rewritten by another rule or removed in (p2). That way it can avoid more closure computation than `(lcm)` at cost of an inexpensive check of attribute presence (Algorithm 6, line 13).

Indeed, our experimental comparison shows that LinCbO with `(lcmx)` performs slightly better than `(lcm)` in most cases (Table 4) and avoids more closure computation (Table 5). However, the difference in the performance is not significant.

Comparison with other algorithms

The column 'best of the rest' represents the best algorithm from the Bazhanov & Obiedkov's framework. We tested all seven algorithms listed above, however only Ganter, Ganter$^+$ (both with naïve closure implementation) and the attribute incremental approach appear in the column, as these performed best in our evaluation. Among these algorithms, the attribute incremental approach was ofthen the fastest one. In some cases, it was even faster than LinCbO without pruning. However, we encountered limits of this algorithm as it runs out of available memory in three cases: `inter10shuttle`, `inter4magic`, and `nom5keg`.

Table 5: Number of skipped recursive calls and speed-up factor by pruning techniques

Dataset	(lcm)	speed-up factor (%)	(lcmx)	speed-up factor (%)
inter10crx	120,851,019	227.66	126,403,951	255.41
inter10shuttle	1,321,766,518	105.91	1,326,688,040	106.93
inter3magic	1,538,199	107.69	1,637,367	108.12
inter4magic	48,536,834	125.19	52,180,055	115.57
inter5bike_day	18,193,052	194.47	19,432,953	212.13
inter5crx	2,345,689	171.21	2,429,752	176.25
inter5shuttle	7,536,887	107.11	7,603,108	106.54
inter6shuttle	9,922,755	105.03	10,029,964	105
nom10bike_day	1,195,268	422.08	1,229,644	459.46
nom10crx	635,844	284.38	641,138	287.87
nom10magic	2,974,506	214.57	2,995,995	213.94
nom10shuttle	39,864	111.05	40,288	107.34
nom15magic	10,129,231	271.05	10,185,502	277.31
nom20magic	19,659,598	366.2	19,756,910	366.56
nom5bike_day	502,879	266.27	533,577	276.04
nom5bike_hour	16,430,989	295.87	17,011,991	293.02
nom5crx	169,499	169.24	171,668	181.19
nom5keg	226,578,200	173.46	227,020,735	169.91
nom5shuttle	12,983	103.91	13,338	105.71
ord10bike_day	2,468,278	156.75	2,848,811	165.45
ord10crx	1,621,895	110.82	2,169,367	114.85
ord10shuttle	1,144,851	94.51	1,181,005	94.71
ord5bike_day	121,968	131.32	156,400	139.91
ord5bike_hour	1,122,408	117.27	1,677,745	124.44
ord5crx	137,169	109.12	161,173	110.74
ord5magic	491,174	97.01	493,737	97.02
ord5shuttle	38,877	98.02	40,987	97.75
ord6magic	1,856,194	99.85	1,867,038	97.28

3 LCM is well implemented CbO

LCM (**L**inear time **C**losed itemset **M**iner) is an algorithm for the enumeration of frequent closed itemsets developed by Takeaki Uno [86, 85, 87, 88] in 2003–2005. It is considered to be one of the most efficient algorithms for this task. Its implementations with source codes are available at `http://research.nii.ac.jp/~uno/codes.htm`. Frequent closed itemsets in transaction databases are exactly intents in formal concept analysis (FCA) with sufficient support—cardinality of the corresponding extents. If the minimum required support is zero (i.e. any attribute set is considered frequent), one can easily unify these two notions.

In this section, we describe LCM in terms of FCA and reveal that LCM is basically the Close-by-One algorithm with multiple speed-up features for processing sparse data.

We have thoroughly studied Uno's papers and source codes and, in this section, we deliver a complete description of LCM from the point of view of FCA. Despite the source codes being among the main sources for this study, we stay at a very comprehensible level in our description and avoid delving into implementation details. We explain that the basis of LCM is Kuznetsov's Close-by-One (CbO) [62].[8] We describe its additional speed-up features and compare them with those of state-of-art CbO-based algorithms, like FCbO [73] and In-Close2+ [6, 7, 8, 9].[9]

There are three versions of the LCM algorithm:

LCM1 is CbO with arraylist representation of data and computing of all extents at once (described in Section 3.2), data preprocessing (described in Section 3.1), and using of diffsets [93] to represent extents for dense data (this is not present in later versions).

LCM2 is LCM1 (without diffsets) with conditional databases (described in Section 3.3)

LCM3 is LCM2 which uses a hybrid data structure to represent a context. The data structure uses a combination of FP-trees and bitarrays, called a complete FP-tree, to handle the most dense attributes. Arraylists are used for the rest, the same way as in the previous versions.

In this paper, we describe all features present in LCM2.

[8] Although LCM was most likely developed independently.

[9] In the rest of this section, whenever we write 'CbO-based algorithms' we mean CbO, FCbO and In-Close family of algorithms. By version number 2+, we mean the version 2 and higher.

3.1 Initialization

To speed the computation up, LCM initializes the input data as follows:

- removes empty rows and columns,
- merges identical rows,
- sorts attributes by cardinality ($|y^{\downarrow}|$) in the descending order,
- sorts objects by cardinality ($|x^{\uparrow}|$) in the descending order.

In the pseudocode in Algorithm 7, the initialization is not shown and it is supposed that it is run before the first invocation of the procedure `GenerateFrom`.

FCA aspect: The attribute sorting is well known to most likely cause a smaller number of computations of closures in CbO-based algorithms [55, 6, 7]. This feature is included in publicly available implementations of In-Close4 and FCbO.

The object sorting is a different story. Andrews [6] tested the performance of In-Close2 and concluded that lexicographic order tends to significantly reduce L1 data cache misses. However, the test were made for bitarray representation of contexts.

The reason for object sorting in LCM is probably that a lesser amount of inverses occurs in a computation of a union of rows (shown later (16)), which is consequently easier to sort. Our testing with Uno's implementation of LCM did not show any difference in runtime for unsorted and sorted objects when attributes are sorted. In the implementation of LCM3, the object sorting is not present.

Remark 5. *In examples in this paper, we do not use sorted data, in order to keep the examples small.*

3.2 Ordered arraylists and occurrence deliver

LCM uses arraylists[10] as data representation of the rows of the context. It is directly bound to one of the LCM's main features – *occurrence deliver*:

LCM computes extents $A \cap i^{\downarrow}$ (line 3 in Algorithm 1) all at once using a single traversal through the data. Specifically, it sequentially traverses through all rows $x^{\uparrow}$ of the context and whenever it encounters an attribute i, it adds x to an initially empty arraylist – *bucket* – for i (see Fig. 6). As LCM works with conditional datasets (see Section 3.3), attribute extents correspond to extents $A \cap i^{\downarrow}$ (see Algorithm 1, line 3). This is also known as *vertical format* in DM algorithms; the buckets are also known as *tidlists*.

[10] Whenever we write arraylist, we mean ordered arraylists.

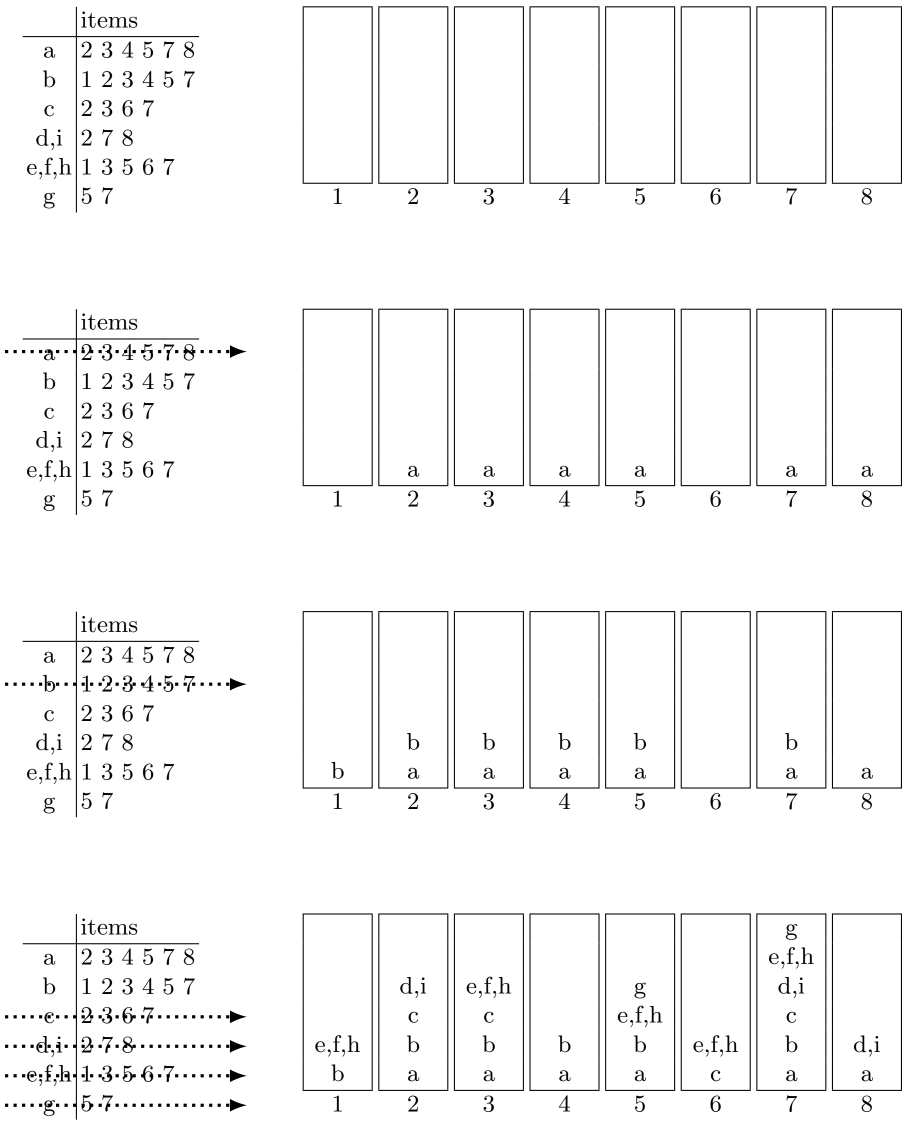

Figure 6: Occurrence deliver in LCM.

LCM generates children of each node from right to left. That way, it can reuse the memory for extents (buckets). For example, when computing extents in the node $\{2\}$, that is $\{2,3\}^{\downarrow}$ and $\{2,4\}^{\downarrow}$, the algorithm can reuse the memory used by extents $\{3\}^{\downarrow}$ and $\{4\}^{\downarrow}$, because $\{3\}$ and $\{4\}$ (and their subtrees) are already finalized.

FCA aspect: In FCA, the CbO-based algorithms do not specify data representation used for handling contexts, sets of objects, and sets of attributes. This is mostly considered a matter of specific implementations (see Remark 6). Generally, the data representation issues are almost neglected in literature on FCA. The well-known comparison study [67] of FCA algorithms mentioned the need to study the influence of data structures on practical performances of FCA algorithms but it does not pay attention to that particular issue. The comparison study [57] provided the first steps to an answer for this need.[11] The latter paper concludes that binary search trees or linked lists are good choices for large or sparse datasets, while bitarray is an appropriate structure for small or dense datasets. Arraylists did not perform particularly well in any setting. However, this comparison did not assume other features helpful for this data representation, like conditional databases (see Section 3.3) and computation of all required attribute extents in one sweep by occurrence deliver. More importantly, the minimal tested density is 5 %, which is still very dense in the context of transactional data.

Remark 6. *Available implementations of FCbO[12] and In-Close[13] utilize bitarrays for rows of contexts, and sets of attributes, and arraylists for sets of objects.*

3.3 Conditional database and interior intersections

LCM reduces the database for the recursive invocations of `GenerateFrom`.

Let $\mathcal{K} = \langle X, Y, I\rangle$ be a formal context, $D \subseteq Y$ be an attribute set which occurred as $D = (B \cup \{i\})^{\downarrow\uparrow}$.

The conditional context $\mathcal{K}_{B,i}$ w.r.t. $\langle B, i\rangle$ is created from $\mathcal{K}$ as follows:

(a) First remove from $\mathcal{K}$ objects which are not in the corresponding extent $A = B^{\downarrow}$ (Fig. 7 (a)).

(b) Remove attributes which are full or empty (Fig. 7 (b)).

(c) Remove attributes lesser than i (Fig. 7 (c))[14]

[11] Paper [57] compares bitarrays, sorted linked lists, arraylists, binary search trees, and hash tables.

[12] Available at `http://fcalgs.sourceforge.net/`.

[13] Available at `https://sourceforge.net/projects/inclose/`.

[14] In the implementation, when the database is already too small (less than 6 objects, and less than 2 attributes), steps (c)–(d) are not performed.

(d) Merge identical objects together (Fig. 7 (d))

(e) Put back attributes removed in step (d), incidences are intersections of the corresponding merged rows (Fig. 7 (e)). The part of context added in this step is called an interior intersection.

Alternatively, we can describe conditional databases with interior intersections as:

- Restricting the context $\mathcal{K}$ to objects in A and attributes in N where

$$N = \left(\bigcup_{x \in A} x^{\uparrow} \right) \setminus A^{\uparrow}. \tag{16}$$

  This covers the steps (a)–(c).

- Subsequent merging/intersecting those objects which have the same incidences with attributes in $\{1, 2, \ldots, y-1\}$. This covers the steps (d)–(e).

In the pseudocode in Algorithm 7, the creation of the conditional databases with interior intersections is represented by procedure named `CreateConditionalDB`$(\mathcal{K}, A, N, y)$.

FCA aspect: CbO-based algorithms do not utilize conditional databases. However, we can see partial similarities with features of CbO-based algorithms.

First, all the algorithms skip attributes work only with part of the formal context given by $B^{\downarrow}$ and $Y \setminus B$. That corresponds to the step (a) and the first part of step (b) (full attributes).

Second, the removal of empty attributes in step (b) utilizes basically the same idea as in In-Close4 [8]: if the present extent A and an attribute intent $i^{\downarrow}$ have no common object, we can skip the attribute i in the present subtree. In FCbO and In-Close3, such attribute would be skipped due to pruning (see Section 3.4).

Steps (c)–(e) have no analogy in CbO algorithms.

Description of LCM without pruning

At this moment, we present pseudocode of LCM (Algorithm 7) with above-described features. For now, we will ignore pruning feature. As in the case for CbO, the algorithm is given by recursive procedure `GenerateFrom`. The procedure takes four arguments: an extent A, a set of attributes B, the last attribute y added to B, and a (conditional) database $\mathcal{K}$). The procedure performs the following steps:

	1	2	3	4	5	6	7	8
a		×	×	×	×		×	
b	×	×	×	×	×		×	
c		×	×			×	×	
d		×					×	×
e	×		×		×	×	×	
f	×	×	×		×	×	×	
g					×		×	
h			×		×	×	×	
i		×						×

(a) Remove objects not in $\{3,7\}^{\downarrow}$:
Objects d,g, and i are removed.

	1	2	3	4	5	6	7	8
a		×	×	×	×		×	
b	×	×	×	×	×		×	
c		×	×			×	×	
e	×		×		×	×	×	
f	×		×		×	×	×	
h			×		×	×	×	

(b) Remove empty and full attributes:
Attributes 3, 7, and 8 are removed.

	1	2	4	5	6
a		×	×	×	
b	×	×	×	×	
c		×			×
e	×			×	×
f	×			×	×
h				×	×

(c) Remove attributes lesser than 3:
Attributes 1 and 2 are removed.

	4	5	6
a	×	×	
b	×	×	
c			×
e		×	×
f		×	×
h		×	×

(d) Merge identical objects together.

	4	5	6
a,b	×	×	
c			×
e,f,h		×	×

(e) Add interior intersections.

	1	2	4	5	6
a,b		×	×	×	
c		×			×
e,f,h				×	×

Figure 7: Obtaining contitional context $\mathcal{K}_{B,y}$ for attribute set $B = \{3, 7\}$ and attribute $y = 3$.

(line 1)	The set N (16) of non-trivial attributes is computed.
(line 2)	The frequencies of all attributes in N are computed, this is made by a single traversal through $\mathcal{K}$ similar to the occurence deliver (described in Section 3.2).
(lines 4–6)	The loop checks whether any attribute in N lesser than y has frequency equal to $\|A\|$. If so, the attribute causes the canonicity test to fail, therefore we end the procedure.
(lines 7–10)	The loop closes B (and updates N) based on the computed frequencies.
(line 11)	As the canonicity is checked and B is closed, the pair $\langle A, B\rangle$ is printed out.
(line 12)	The conditional database $\mathcal{K}_{B,y}$ (described in Sec. 3.3) is created.
(line 13)	Attribute extents from $\mathcal{K}_{B,y}$ are computed using occurence deliver (described in Section 3.2).
(lines 14, 16)	The procedure `GenerateFrom` is recursively called for attributes in N with the conditional database $\mathcal{K}_{B,y}$ and the corresponding attribute extent.

3.4 Bonus feature: pruning

The jumps using closures in CbO significantly reduce the number of visited nodes in comparison with the naïve algorithm. The closure, however, becomes the most time consuming operation in the algorithm. The pruning technique in LCM[15] avoids computations of some closures based on the monotony property: for any set of attributes $B, D \subseteq Y$ satisfying $B \subseteq D$, we have

$$j \in (B \cup \{i\})^{\downarrow\uparrow} \quad \text{implies} \quad j \in (D \cup \{i\})^{\downarrow\uparrow}. \tag{17}$$

When $i, j \notin D$ and $j < i$, the implication (17) says that if j causes $(B\cup\{i\})^{\downarrow\uparrow}$ to fail the canonicity test then it also causes $(D\cup\{i\})^{\downarrow\uparrow}$ to fail the canonicity test. That is, if we store that $(B \cup \{i\})^{\downarrow\uparrow}$ failed, we can use it to skip computation of the closure $(D \cup \{i\})^{\downarrow\uparrow}$ for any $D \supset B$ with $j \notin D$. We demonstrate this in the following example.

Example 3. *Consider the following formal-context.*

	1	2	3	4
a	×	×		×
b	×		×	×
c		×	×	
d	×			

[15] Pruning is not described in papers on LCM, however, it is present in the implementation of LCM2.

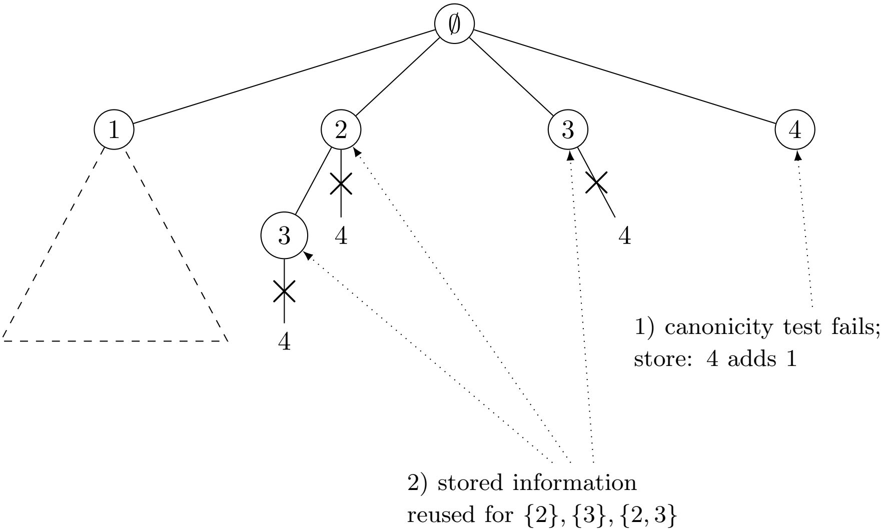

Figure 8: Reuse of a failed canonicity test information

Consider the tree of all subsets in Fig. 4. The rightmost branch of the tree represents adding the attribute 4 *into an empty set. We can easily see, that*

$$\{4\}^{\downarrow\uparrow} = \{1, 4\}, \tag{18}$$

and, therefore, the canonicity test $B_i = D_i$ *fails. In this case, we have* $B_i = \emptyset_4 = \emptyset$ *while* $D_i = \{1, 4\}_4 = \{1\}$.

Notice, that (18) *gives us information for the actual set (an empty set in this case): adding attribute* 4 *causes that attribute* 1 *is in the closed set. Due to* (17) *this holds true for any superset of the actual set. This information is then reused for the supersets. Specifically, for sets* $\{2\}, \{3\}$*, or* $\{2, 3\}$*, adding attribute* 4 *causes that attribute 1 is present in the closed set and, consequently, causes failing the canonicity test. Figure 8 shows the described situation.*

LCM utilizes the above idea in the following way:

(p0) Whenever the canonicity test fails for $(B \cup \{i\})^{\downarrow\uparrow}$ and j is the smallest attribute in $(B \cup \{i\})^{\downarrow\uparrow} \setminus B$, we store the rule "$i$ adds j". In the pseudocode (Algorithm 7) this is realized through the return value of the procedure `GenerateFrom`. The procedure returns the least attribute which caused the canonicity to fail (line 6) or 0 if it passed the canonicity test (line 20). The returned value is used to form a pruning rule to be stored (lines 17,18).

(p1) At the beginning of `GenerateFrom`, i.e. when descending to a subtree, all rules having the last added attribute (argument y) on the right side are removed from the stored rules. In the pseudocode, this is realized by a subroutine called `RemoveRulesByRightSide` (line 3).

(p2) At the end of `GenerateFrom`, i.e. when backtracking from the current subtree, all rules from this call are removed. In the pseudocode, this is realized by a subroutine called `RemoveAllRulesAddedThisCall` (line 19).

(p3) Before computing a closure $(D \cup \{i\})^{\downarrow\uparrow}$ in a subtree of B, we check the stored rules to find whether adding i does not add any attribute which causes the canonicity test to fail. Due to the way how the rules are handled in the previous items, (p0)–(p2), it is sufficient to check whether there is any rule having i on the left side. In the pseudocode, this is realized by a subroutine called `CheckRulesByLeftSide` (line 15).

FCA aspect: Similar pruning techniques are also present in FCbO and In-Close3 and higher:

- FCbO, In-Close3: stores rules of the form "i gives set A".
- In-Close4: stores rules of the form "i gives empty extent".
- In-Close5: stores rules of the form "i adds an attribute which makes the canonicity test fail" and rules of In-Close4.

All the FCA algorithms utilize only steps (p0), (p2), and (p3); none of them performs (p1).

LCM's pruning is weaker than the pruning in FCbO and In-Close3, stronger than the pruning in In-Close4 and In-Close5:

$$\text{In-Close4} < \text{In-Close5} < \text{LCM} < \text{FCbO} = \text{In-Close3}.$$

4 Interface between logical analysis of data and formal concept analysis

While logical analysis of data (LAD) and formal concept analysis (FCA) stands on different mathematical foundations (boolean functions and combinatorics in the case for LAD, lattice theory and closure structures in the case for FCA), there is a link between formal concepts of FCA and patterns of LAD based on the equivalence of their basic building blocks. We see this equivalence to be an interface between FCA and LAD as it enables us to transfer theorems and algorithms from one methodology to the other.

Algorithm 7: LCM

```
def GenerateFrom(A, B, y, 𝒦):
    input : A – extent
            B – set of attributes
            y – last added attribute
            𝒦 – conditional database

    N ← (⋃_{x∈A} x↑) \ B
    {n_i | i ∈ N} ←Frequencies(𝒦, N)

    RemoveRulesByRightSide(y)

    for i ∈ N, i < y do
        if n_i = |A| then
            return i

    for i ∈ N, i > y do
        if n_i = |A| then
            B ← B ∪ {i}
            N ← N \ {i}

    print(⟨A, B⟩)

    𝒦′ ← CreateConditionalDB(𝒦, A, N, y)
    {C_i | i ∈ N} ← OccurenceDeliver(𝒦′)

    for i ∈ N, i > y, (in descending order) do
        if CheckRulesByLeftSide(i) then
            j ← GenerateFrom(C_i, B ∪ {i}, i, 𝒦′)
            if j > 0 then
                AddRule (“i adds j”)

    RemoveAllRulesAddedThisCall()
    return 0

GenerateFrom(X, X↑, 0, ⟨X, Y, I⟩)
```

In this section we describe the link between the methodologies, its main benefits, and areas to be subsequently studied in our future research. The link is based on the three following ideas:

Idea 1: We can consider Ω^+ and Ω^- to be two two-valued formal contexts. We denote by $Y = \{1, \ldots, n\}$ the LAD's set of attributes, by $\Omega = \Omega^+ \cup \Omega^-$ the set of all observations, and by $I_\Omega \subseteq \Omega \times Y$ the relation describing incidences between the observations in Ω and the attributes in Y. That is, for $\omega \in \Omega, y \in Y$ (note, that ω is technically a set; see Remark 3), we have

$$\langle \omega, y \rangle \in I_\Omega \quad \text{iff} \quad y \in \omega. \tag{19}$$

In what follows, we consider the formal context $\langle \Omega, Y, I_\Omega \rangle$ and its restrictions $\langle \Omega^+, Y, I_\Omega^+ \rangle$, $\langle \Omega^-, Y, I_\Omega^- \rangle$ to positive and negative observations. We call the two restrictions positive and negative context, respectively. The concept-forming operators induced by $\langle \Omega, Y, I_\Omega \rangle$, $\langle \Omega^+, Y, I_\Omega^+ \rangle$, and $\langle \Omega^-, Y, I_\Omega^- \rangle$ are respectively denoted by $\langle {}^\vartriangle, {}^\triangledown \rangle$, $\langle {}^{\vartriangle_+}, {}^{\triangledown_+} \rangle$, and $\langle {}^{\vartriangle_-}, {}^{\triangledown_-} \rangle$.

Idea 2: Considering $\langle \Omega^+, \Omega^- \rangle$ a two-valued context, we have

$$\mathrm{Cov}(\boldsymbol{B}) = \boldsymbol{B}^\triangledown.$$

That is, the coverage corresponds to the concept-forming operator ${}^\triangledown$.

Idea 3: We have

$$T^\vartriangle = \mathrm{Span}(T).$$

where ${}^\vartriangle$ is the concept-forming operator induced by $\langle \Omega, Y, I_\Omega \rangle$ (see Idea 1).

4.1 Spanned patterns

First, we can straightforwardly declare a relationship between spanned intervals and formal concepts.

Theorem 2.

(a) Spanned intervals are exactly intents in $\mathrm{Int}(I_\Omega)$.

(b) Intervals spanned by subsets of Ω^+ *are exactly intents in* $\mathrm{Int}(I_\Omega^+)$.

(c) Intervals spanned by subsets of Ω^- *are exactly intents in* $\mathrm{Int}(I_\Omega^-)$.

Now we only need to filter out those intervals $\boldsymbol{B}$ which are not patterns, that is those which satisfy

$$\boldsymbol{B} \cap \Omega^- \neq \emptyset \tag{20}$$

in the case for positive patterns, and

$$\boldsymbol{B} \cap \Omega^{+} \neq \emptyset \tag{21}$$

in the case for negative patterns.

Let us denote the set of formal concepts in $\mathcal{B}(I)$ whose intents are also positive patterns by $\mathcal{B}^{+}(I)$. The corresponding set of intents is denoted by $\mathrm{Int}^{+}(I)$. Analogously, for negative patterns, we use the notations $\mathcal{B}^{-}(I)$ and $\mathrm{Int}^{-}(I)$.

Theorem 3. *We have*

$$\begin{aligned} \mathrm{SPAN}^{+}(\Omega) &= \mathrm{Int}^{+}(I_{\Omega}^{+}), \\ \mathrm{SPAN}^{-}(\Omega) &= \mathrm{Int}^{-}(I_{\Omega}^{-}). \end{aligned}$$

Using the above remarks, we can compute spanned patterns using an algorithm for enumeration of formal concepts (or just intents), filtering out those which are not patterns.

4.2 Prime and strong patterns

In this section, we characterize prime and strong patterns in terms of FCA. First, we need to recall the notion of a generator.

Definition 1. A generator of an intent $\boldsymbol{B} \in \mathrm{Int}(I)$ is an interval $\boldsymbol{C} \in \mathcal{I}^{Y}$ such that

$$\boldsymbol{C}^{\nabla\Delta} = \boldsymbol{B}.$$

Clearly, it must hold that $\boldsymbol{B} \subseteq \boldsymbol{C}$. If there is no generator $\boldsymbol{D}$ of $\boldsymbol{B}$ such that

$$\boldsymbol{B} \subseteq \boldsymbol{C} \subset \boldsymbol{D},$$

we call $\boldsymbol{C}$ a maximal generator of $\boldsymbol{B}$.

Now we can provide the following characterizations of prime and strong patterns.

Theorem 4. *Strong positive patterns are exactly generators of maximal elements of* $\mathrm{Int}^{+}(I_{\Omega}^{+})$. *Strong negative patterns are exactly generators of maximal elements of* $\mathrm{Int}^{-}(I_{\Omega}^{-})$.

Proof. Take any maximal concept $\langle A, \boldsymbol{B}\rangle$ of $\mathcal{B}^{+}(I_{\Omega}^{+})$. Consider $x \in \Omega^{+}$ such that $x \notin A$. Note that there is no interval $\boldsymbol{C}$ such that

$$\boldsymbol{C}^{\nabla_{+}} \supseteq \{x\} \cup A.$$

Indeed, if there is such $\boldsymbol{C}$, then $\langle \boldsymbol{C}^{\nabla+}, \boldsymbol{C}^{\nabla+\Delta+} \rangle$ is a formal concept in $\mathcal{B}^+(I_\Omega^+)$ and $\langle A, \boldsymbol{B} \rangle$ is not maximal. Therefore, there is no interval $\boldsymbol{C} \geq \boldsymbol{B}$ such that $\boldsymbol{C}^{\nabla+} \supset A$. Any $\boldsymbol{C} \geq \boldsymbol{B}$ satisfying $\boldsymbol{C}^{\nabla+} = A$ is a strong positive pattern.

Let $\boldsymbol{C}$ be a strong positive pattern. Clearly, $\langle \boldsymbol{C}^{\nabla+}, \boldsymbol{C}^{\nabla+\Delta+} \rangle \in \mathcal{B}^+(I_\Omega^+)$ and $\boldsymbol{C}$ is a generator of $\boldsymbol{C}^{\nabla+\Delta+}$. By definition of a positive pattern, there is no interval $\boldsymbol{D}$ such that $\boldsymbol{D}^{\nabla+} \supset \boldsymbol{C}^{\nabla+}$. That means that $\boldsymbol{C}^{\nabla+}$ is an extent of a maximal concept in $\mathcal{B}^+(I_\Omega^+)$.

Analogously for negative patterns. $\square$

Theorem 5. *Prime positive patterns are exactly maximal generators of maximal elements of* $\mathrm{Int}^+(I_\Omega^+)$. *Prime negative patterns are exactly maximal generators of maximal elements of* $\mathrm{Int}^-(I_\Omega^-)$.

Proof. Directly from Theorem 4. $\square$

Considerable research on minimal generators has been done in FCA [82, 83, 71]; see also related sections in surveys [75, 76].

4.3 Selected benefits of the interface

The proposed interface between the two methodologies has a potential to bring fruitful results. In this section, we describe the three most obvious and present results of our preliminary experiments.

4.3.1 Efficient algorithms of FCA applicable in LAD

Algorithms for computing spanned patterns

Literature on LAD [3, 1, 26] describes two algorithms to generate spanned patterns called SPAN and SPIC. To describe them, we need to introduce the notion of consensus. Let $\boldsymbol{B}_1 = [\underline{B}_1, \overline{B}_1]$, $\boldsymbol{B}_2 = [\underline{B}_2, \overline{B}_2]$ be two spanned patterns. If the interval

$$\boldsymbol{B}_1 \sqcup \boldsymbol{B}_2 = [\underline{B}_1 \cap \underline{B}_2, \overline{B}_1 \cup \overline{B}_2]$$

is a pattern, we call it the *consensus* of the two patterns.

The algorithm SPAN corresponds to what Ganter and Wille [36] call a naïve approach in FCA. It starts with observations in Ω^+ and generates new spanned patterns as consensus of already found patterns pairwise. The algorithm terminates when no two spanned patterns produce a new spanned pattern as their consensus.

Algorithm SPIC is a variant of SPAN which avoids some computations which lead to duplicated patterns. Specifically, one of the two patterns to make a consensus has to be an observation from Ω^+. In FCA, this corresponds to the algorithm *Object Intersections* described in [23]. For a detailed description, see Appendix B in [48].

When we write that SPIC and SPAN correspond to particular algorithms in FCA in the above paragraphs, we mean that we can generate spanned positive patterns as elements of $\mathrm{Int}^+(I_\Omega^+)$ with the algorithms modified to the two-valued setting (analogously for negative patterns).

Experiments

To support our claims on the efficiency of FCA algorithms used for computation of LAD's patterns, we performed some preliminary experiments. Using algorithms CbO, FCbO and SPIC, we computed the first 1000, 5000, 10000, 15000, and 20000 positive spanned patterns with 5% prevalence (percentage of covered observations) in four datasets from the UC Irvine Machine Learning Repository [32], namely *Breast Cancer Wisconsin (bcw)*, *Mushrooms*, *Tic-Tac-Toe*, and *Congressional Voting Records (votes)*.

All three algorithms were implemented in C++, sharing a common code base and data structures. Namely, bit-vectors were used to represent patterns and intents. Note that this representation allows for efficient implementation of the intersection operation which is essential for all discussed algorithms.

Our experiments were performed on a computer equipped with 64 GB RAM, two Intel Xeon E5-2680 CPUs, 2.80 GHz, and Debian Linux 9.6 with GNU GCC 6.3.0.

We measured the runtime required to finish the task. All measurements were taken three times and an average value was used. In all cases, the time required by the FCA algorithms was several orders of magnitude less than the time required by SPIC; see Table 6.

Remark 7 (Space complexity)**.** *We did not compare the memory used by the algorithms. We only comment on asymptotic space complexity of the algorithms. SPIC needs to keep generated patterns in the memory to check for duplicates. This leads to an exponential space complexity as, in the worst case,* $\mathcal{O}(2^{|Y|})$ *patterns are stored in the memory. In contrast, FCA algorithms assure uniqueness of each enumerated pattern and their space complexity is in* $\mathcal{O}(|Y|^2)$.

Remark 8. *It is important to note that the SPIC algorithm and the two FCA algorithms enumerate spanned patterns in a different order. Therefore, the first 1000 spanned patterns computed by SPIC and the first 1000 spanned patterns computed by CbO or FCbO are different sets of patterns. Due to this difference, we cannot simply conclude superiority of the FCA algorithms. This is why we call the experiments preliminary. Further study in this area is needed and is planned for our future research.*

dataset	alg.	1000	5000	10000	15000	20000
bcw	CbO	11	49	87.91	122	153
	FCbO	10	40	72.50	101	129
	SPIC	184	667	1124.63	1299	1594
mushrooms	CbO	231	800	1336.74	2033	2596
	FCbO	64	165	273.55	389	486
	SPIC	3181330	6023211	12094900	19057156	24869203
tic-tac-toe	CbO	599	1412	2207.90	3287	3811
	FCbO	210	447	720.49	978	1122
	SPIC	3002	5653	10803	18505	32427
votes	CbO	11	51	92.80	126	162
	FCbO	5	16	28.67	47	60
	SPIC	208	928	1269.10	1631	1912

Table 6: Comparison of running time (in milliseconds) required for computation of the first one thousand, five thousand, ..., twenty thousand spanned patterns with prevalence 5 % using CbO, FCbO and SPIC.

4.3.2 Concept rankings and reductions of concept lattices

One of the most recognized problems in both LAD and FCA is that a very large amount of patterns/formal concepts can be generated from the input data. Despite the understandability of the patterns and formal concepts, the large quantity becomes ungraspable and unreadable by a human user. Additionally, the large quantity is unfeasible for further processing. In LAD, we need to select a *model*, a representative subset of patterns for classification. The main emphasis is on covering all observations and, additionally, on handling outliers [43]. The selection of the model is part of a process called *theory formation*.

In FCA, many studies are devoted to the reduction of the size of a concept lattice; see survey studies [30, 31]. Furthermore, multiple studies in FCA considered various measures of relevancy of formal concepts. For instance, *stability* [61, 65, 60] and *basic level* [16, 17]. The comparative study [66] provides a comparison of relevancy measures of formal concepts with respect to various aspects. We plan to perform an experimental evaluation of reductions and relevancy measures with respect to the goals of LAD.

4.3.3 Generalization to graded setting

In the real world, incidences between observations and attributes are rarely a matter of absolute truth and absolute falsity. Rather, it is a matter of degrees of truth (like ‘almost true’, ‘more or less false’, etc.) Formal concept analysis was generalized to handle these degrees of truth in the 90s independently by Belohlavek [14] and Burusco [22]. This generalization is based on the framework of **L**-fuzzy sets [37, 38] known as Formal Fuzzy Concept Analysis (FFCA).

FFCA was further generalized to enable us to process positive and negative attributes [10, 11]. The interface described in the present paper can be almost directly used to design a generalization of LAD for handling degrees of truth. We see this as a key part of our future research.

5 Conclusions

In this paper we brought three following results:

- We introduced the new algorithm called LinCbO for computation of the Duquenne-Guigues basis. It uses natural behavior of the Close-by-One algorithm for speed-up of a computation. We can use values of the attribute counters from previous calls of LinClosure, so subsequent calls of LinClosure are faster. We also equipped LinCbO with pruning techniques to avoid some unnecessary recursive calls.

  We demonstrated this speed-up feature on experiments with real and artificial datasets. We showed that the LinCbO algorithm is very fast and has great potential for further research and improvements.

- We described the LCM algorithm from the FCA point of view. Formerly, we used it as a black box. Now we know that it shares basic ideas with the CbO. In the available implementation of LCM we also discovered the pruning technique, which was not described in original papers. We implemented its pruning technique into LinCbO; detailed comparison with the other pruning techniques will be a subject of our future research.

- We stated the interface between formal concept analysis and logical analysis of data and we showed that it could bring some benefits. Namely, we can use efficient algorithms from FCA for enumerating all patterns. In future work, we will investigate the benefits of this interface. We hope that the two methodologies can enrich each other.

We wanted to show the Close-by-One algorithm in a new light. We showed that this well-known algorithm still has potential for improvements and can be used in new fields.

Shrnutí v českém jazyce

V této práci jsme přinesli následující výsledky:

- Představili náš nový algoritmus LinCbO pro výpočet Duquenne-Guigues báze. Pro zrychlení výpočtu využívá přirozených vlastností algoritmu Close-by-One, díky kterým můžeme znovu využít hodnoty atributových čítačů v algoritmu LinClosure. To vede k výraznému urychlení dalších volání LinClosure. Dále jsme LinCbO vylepšili pomocí technik prořezávání, díky kterým je možné vynechat některá rekurzivní volání.

  Provedli jsme experimenty s reálnými i uměle generovanými datasety, na kterých jsme demonstrovali dopad našich vylepšení. Na experimentech jsme ukázali, že LinCbO je velmi rychlý a má velký potenciál k budoucímu zkoumání.

- Ukázali jsme, jak funguje algoritmus LCM z pohledu formální konceptuální analýzy. Tento algoritmus jsme dříve používali jako *černou skříňku*, dnes víme, že sdílí základní myšlenky s algoritmem CbO. V dostupné implementaci LCM jsme objevili, že je také použito přeřetávání, které nebylo popsáno v původních článcích. Stejnou metodu prořezávání jsme implementovali i do LinCbO. Detailní srovnání s ostatními technikami prořezávání bude předmětem našeho dalšího zkoumání.

- V poslední části jsme popsali rozhraní mezi formální konceptuální analýzou a logickou analýzou dat a ukázali jsme, že může přinést mnoho výhod. Například, můžeme použít efektivní algoritmy z FCA pro výpočet všech *vzorů*(patterns). V budoucnu chceme podrobněji prozkoumat všechny výhody tohoto rozhraní. Věříme, že se tyto metodiky mohou navzájem obohatit.

  V této práci jsme chtěli ukázat nový pohled na algoritmus Close-by-One. Ukázali jsme, že tento známý algoritmus má stále potenciál pro vylepšování a může být použit v nových oblastech.

References

[1] G. Alexe, S. Alexe, T. O. Bonates, and A. Kogan. Logical Analysis of Data–the vision of Peter L. Hammer. *Annals of Mathematics and Artificial Intelligence*, 49(1-4):265–312, 2007.

[2] G. Alexe, S. Alexe, P. L. Hammer, and A. Kogan. Comprehensive vs. comprehensible classifiers in Logical Analysis of Data. *Discrete Applied Mathematics*, 156(6):870–882, 2008.

[3] G. Alexe and P. L. Hammer. Spanned patterns for the Logical Analysis of Data. *Discrete Applied Mathematics*, 154(7):1039–1049, 2006.

[4] S. Alexe, E. Blackstone, P. L. Hammer, H. Ishwaran, M. S. Lauer, and C. E. Pothier Snader. Coronary risk prediction by Logical Analysis of Data. *Annals of Operations Research*, 119(1):15–42, 2003.

[5] S. Andrews. In-Close, a fast algorithm for computing formal concepts. In *International Conference on Conceptual Structures*. Springer, 2009.

[6] S. Andrews. In-Close2, a high performance formal concept miner. In *Proceedings of the 19th International Conference on Conceptual Structures for Discovering Knowledge*, pages 50–62. Springer-Verlag, 2011.

[7] S. Andrews. A 'Best-of-Breed' approach for designing a fast algorithm for computing fixpoints of Galois connections. *Information Sciences*, 295:633–649, 2015.

[8] S. Andrews. Making use of empty intersections to improve the performance of CbO-type algorithms. In *International Conference on Formal Concept Analysis*, pages 56–71. Springer, 2017.

[9] S. Andrews. A new method for inheriting canonicity test failures in Close-by-One type algorithms. In *CLA 2018: The 14th International Conference on Concept Lattices and Their Applications*, pages 255–266. Springer, 2018.

[10] E. Bartl and J. Konecny. L-concept analysis with positive and negative attributes. *Information Sciences*, 360:96–111, 2016.

[11] E. Bartl and J. Konecny. L-concept lattices with positive and negative attributes: Modeling uncertainty and reduction of size. *Information Sciences*, 472:163–179, 2019.

[12] K. Bazhanov and S. A. Obiedkov. Optimizations in computing the Duquenne-Guigues basis of implications. *Annals of Mathematics and Artificial Intelligence*, 70(1-2):5–24, 2014.

[13] C. Beeri and P. A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems (TODS)*, 4(1):30–59, 1979.

[14] R. Belohlavek. *Fuzzy relational systems: foudations and principles*. IFSR International Series on Systems Science and Engineering Series. Kluwer Academic/Plenum Publishers, 2002.

[15] R. Belohlavek, E. Sigmund, and J. Zacpal. Evaluation of IPAQ questionnaires supported by Formal Concept Analysis. *Information Sciences*, 181(10):1774–1786, 2011.

[16] R. Belohlavek and M. Trnecka. Basic level of concepts in Formal Concept Analysis. In *Formal Concept Analysis*, pages 28–44. Springer, 2012.

[17] R. Belohlavek and M. Trnecka. Basic level in Formal Concept Analysis: Interesting concepts and psychological ramifications. In *International Joint Conference on Artificial Intelligence*, pages 1233–1239, 2013.

[18] R. Belohlavek and V. Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences*, 76(1):3–20, 2010. Special Issue on Intelligent Data Analysis.

[19] G. Birkhoff. *Lattice theory*, volume 25. American Mathematical Society, 1940.

[20] E. Boros, P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik. An implementation of Logical Analysis of Data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):292–306, March 2000.

[21] R. Bruni, G. Bianchi, C. Dolente, and C. Leporelli. Logical Analysis of Data as a tool for the analysis of probabilistic discrete choice behavior. *Computers & Operations Research*, 106:191–201, 2019.

[22] A. Burusco and R. Fuentes-Gonzáles. The study of the L-fuzzy concept lattice. *Annals of Pure and Applied Logic*, I(3):209–218, 1994.

[23] C. Carpineto and G. Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, 2004.

[24] C. Carpineto and G. Romano. Using concept lattices for text retrieval and mining. In *Formal Concept Analysis*, pages 161–179. Springer, 2005.

[25] C. Carpineto, G. Romano, and F. U. Bordoni. Exploiting the potential of concept lattices for information retrieval with CREDO. *Journal of Universal Computer Science*, 10(8):985–1013, 2004.

[26] I. Chikalov, V. Lozin, I. Lozina, M. Moshkov, H. S. Nguyen, A. Skowron, and B. Zielosko. *Three Approaches to Data Analysis: Test Theory, Rough Sets and Logical Analysis of Data*, volume 41. Springer, 01 2013.

[27] R. Cole and P. Eklund. Browsing semi-structured web texts using Formal Concept Analysis. In *International Conference on Conceptual Structures*, pages 319–332. Springer, 2001.

[28] R. Cole and G. Stumme. CEM – a conceptual email manager. In *International Conference on Conceptual Structures*, pages 438–452. Springer, 2000.

[29] Y. Crama, P. L. Hammer, and T. Ibaraki. Cause-effect relationships and partially defined boolean functions. *Annals of Operations Research*, 16(1-4):299–325, 1988.

[30] S. M. Dias and N. J. Vieira. Concept lattices reduction: Definition, analysis and classification. *Expert Systems with Applications*, 42(20):7084–7097, 2015.

[31] S. M. Dias and N. J. Vieira. A methodology for analysis of concept lattice reduction. *Information Sciences*, 396:202–217, 2017.

[32] D. Dua and C. Graff. UCI machine learning repository, 2017.

[33] C. Dupuis, M. Gamache, and J.-F. Pagé. Logical Analysis of Data for estimating passenger show rates at Air Canada. *Journal of Air Transport Management*, 18(1):78–81, 2012.

[34] B. Ganter and S. O. Kuznetsov. Formalizing hypotheses with concepts. In *Conceptual Structures: Logical, Linguistic, and Computational Issues*, pages 342–356. Springer, 2000.

[35] B. Ganter and K. Reuter. Finding all closed sets: A general approach. *Order*, 8(3):283–290, 1991.

[36] B. Ganter and R. Wille. *Formal Concept Analysis – Mathematical Foundations*. Springer, 1999.

[37] J. A. Goguen. L-fuzzy sets. *Journal of Mathematical Analysis and Applications*, 18(1):145–174, 1967.

[38] J. A. Goguen. The logic of inexact concepts. *Synthese*, 19(3):325–373, 1969.

[39] J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives resultant d'un tableau de données binaires. *Mathématiques et sciences humaines*, 95:5–18, 1986.

[40] A. B. Hammer, P. L. Hammer, and I. Muchnik. Logical analysis of chinese labor productivity patterns. *Annals of Operations Research*, 87:165–176, 1999.

[41] P. L. Hammer, A. Kogan, and M. A. Lejeune. Modeling country risk ratings using partial orders. *European Journal of Operational Research*, 175(2):836–859, 2006.

[42] P. L. Hammer, A. Kogan, and M. A. Lejeune. Reverse-engineering country risk ratings: a combinatorial non-recursive model. *Annals of Operations Research*, 188(1):185–213, 2011.

[43] J. Han, N. Kim, B.-J. Yum, and M. K. Jeong. Pattern selection approaches for the Logical Analysis of Data considering the outliers and the coverage of a pattern. *Expert Systems with Applications*, 38(11):13857–13862, 2011.

[44] W. Hesse and T. Tilley. Formal Concept Analysis used for software analysis and modelling. In *Formal Concept Analysis*, pages 288–303. Springer, 2005.

[45] A. Hotho, A. Maedche, and S. Staab. Ontology-based text document clustering. *Künstliche Intelligenz*, 16(4):48–54, 2002.

[46] R. Janostik and J. Konecny. General framework for consistencies in decision contexts. *Information Sciences*, 530:180–200, 2020.

[47] R. Janostik, J. Konecny, and P. Krajča. LCM is well implemented CbO: study of LCM from FCA point of view. In *CLA*, pages 47–58, 2020.

[48] R. Janostik, J. Konecny, and P. Krajča. Interface between logical analysis of data and formal concept analysis. *European Journal of Operational Research*, 2020.

[49] R. Janostik, J. Konecny, and P. Krajča. LCM is well implemented CbO: study of LCM from FCA point of view. *CoRR*, abs/2010.06980, 2020.

[50] R. Janostik, J. Konecny, and P. Krajča. LinCbO: fast algorithm for computation of the Duquenne-Guigues basis. *CoRR*, abs/2011.04928, 2020.

[51] M. Kaytoue, S. Duplessis, S. O. Kuznetsov, and A. Napoli. Two FCA-based methods for mining gene expression data. In *Formal Concept Analysis*, pages 251–266. Springer, 2009.

[52] M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining gene expression data with pattern structures in Formal Concept Analysis. *Information Sciences*, 181(10):1989 – 2001, 2011. Special Issue on Information Engineering Applications Based on Lattices.

[53] W. C. Kneale and M. Kneale. *The Development of Logic*. Oxford University Press, USA, 1985.

[54] J. Konecny and P. Krajča. Pruning in Map-Reduce style CbO algorithms. In *Ontologies and Concepts in Mind and Machine*, pages 103–116. Springer, 2020.

[55] P. Krajca, J. Outrata, and V. Vychodil. Advances in algorithms based on CbO. In *Concept Lattices and Their Applications*, volume 672, pages 325–337, 2010.

[56] P. Krajca, J. Outrata, and V. Vychodil. Parallel algorithm for computing fixpoints of Galois connections. *Annals of Mathematics and Artificial Intelligence*, 59(2):257–272, 2010.

[57] P. Krajca and V. Vychodil. Comparison of data structures for computing formal concepts. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 114–125. Springer, 2009.

[58] P. Krajca and V. Vychodil. Distributed algorithm for computing formal concepts using Map-Reduce framework. In *Advances in Intelligent Data Analysis VIII*, pages 333–344. Springer, 2009.

[59] S. Kuznetsov. Machine learning on the basis of Formal Concept Analysis. *Automation and Remote Control*, 62(10):1543–1564, 2001.

[60] S. Kuznetsov, S. Obiedkov, and C. Roth. Reducing the representation complexity of lattice-based taxonomies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4604 LNAI:241–254, 2007.

[61] S. O. Kuznetsov. Stability as an estimate of the degree of substantiation of hypotheses derived on the basis of operational similarity. *Nauchn. Tekh. Inf., Ser.2 (Automatic Documentation and Mathematical Linguistics)*, 12(12):21–29, 1990.

[62] S. O. Kuznetsov. A fast algorithm for computing all intersections of objects from an arbitrary semilattice. *Nauchno-Tekhnicheskaya Informatsiya Seriya 2-Informatsionnye Protsessy i Sistemy*, (1):17–20, 1993.

[63] S. O. Kuznetsov. Complexity of learning in concept lattices from positive and negative examples. *Discrete Applied Mathematics*, 142(1):111 – 125, 2004.

[64] S. O. Kuznetsov. On the intractability of computing the Duquenne-Guigues base. *Journal of Universal Computer Science*, 10(8):927–933, 2004.

[65] S. O. Kuznetsov. On stability of a formal concept. *Annals of Mathematics and Artificial Intelligence*, 49(1-4):101–115, 2007.

[66] S. O. Kuznetsov and T. P. Makhalova. On interestingness measures of formal concepts. *Information Sciences*, 442-443:202–219, 2018.

[67] S. O. Kuznetsov and S. Obiedkov. Paring performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14:189–216, 2002.

[68] M. Lejeune, V. Lozin, I. Lozina, A. Ragab, and S. Yacout. Recent advances in the theory and practice of Logical Analysis of Data. *European Journal of Operational Research*, 275(1):1–15, 2019.

[69] P. Lemaire, N. Brauner, P. Hammer, C. Trivin, J.-C. Souberbielle, and R. Brauner. Improved screening for growth hormone deficiency using Logical Analysis Data. *Medical Science Monitor*, 15(1):MT5–MT10, 2008.

[70] D. Maier. *The theory of relational databases*, volume 11. Computer science press Rockville, 1983.

[71] K. Nehmé, P. Valtchev, M. H. Rouane, and R. Godin. On computing the minimal generator family for concept lattices and icebergs. In *International Conference on Formal Concept Analysis*, pages 192–207. Springer, 2005.

[72] S. Obiedkov and V. Duquenne. Attribute-incremental construction of the canonical implication basis. *Annals of Mathematics and Artificial Intelligence*, 49(1-4):77–99, 2007.

[73] J. Outrata and V. Vychodil. Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. *Information Sciences*, 185(1):114–127, 2012.

[74] J. Poelmans, P. Elzinga, D. I. Ignatov, and S. O. Kuznetsov. Semi-automated knowledge discovery: identifying and profiling human trafficking. *International Journal of General Systems*, 41(8):774–804, 2012.

[75] J. Poelmans, P. Elzinga, S. Viaene, and G. Dedene. Formal Concept Analysis in knowledge discovery: a survey. In *International conference on conceptual structures*, pages 139–153. Springer, 2010.

[76] J. Poelmans, D. I. Ignatov, S. O. Kuznetsov, and G. Dedene. Formal Concept Analysis in knowledge processing: A survey on applications. *Expert systems with applications*, 40(16):6538–6560, 2013.

[77] J. Qi, T. Qian, and L. Wei. The connections between three-way and classical concept lattices. *Knowledge-Based Systems*, 91:143–151, 2016.

[78] T. Qian, L. Wei, and J. Qi. Constructing three-way concept lattices based on apposition and subposition of formal contexts. *Knowledge-Based Systems*, 116:39–48, 2017.

[79] J. M. Rodríguez-Jiménez, P. Cordero, M. Enciso, and A. Mora. Negative attributes and implications in Formal Concept Analysis. *Procedia Computer Science*, 31:758–765, 2014.

[80] J. M. Rodríguez-Jiménez, P. Cordero, M. Enciso, and S. Rudolph. Concept lattices with negative information: A characterization theorem. *Information Sciences*, 369:51–62, 2016.

[81] G. Snelting. Concept lattices in software analysis. In *Formal Concept Analysis*, pages 272–287. Springer, 2005.

[82] G. Stumme. Efficient data mining based on Formal Concept Analysis. In *International Conference on Database and Expert Systems Applications*, pages 534–546. Springer, 2002.

[83] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data & knowledge engineering*, 42(2):189–222, 2002.

[84] T. Tilley and P. Eklund. Citation analysis using Formal Concept Analysis: A case study in software engineering. In *Database and Expert Systems Applications, 2007. DEXA'07. 18th International Workshop on*, pages 545–550. IEEE, 2007.

[85] T. Uno, T. Asai, Y. Uchida, and H. Arimura. LCM: An efficient algorithm for enumerating frequent closed item sets. In *FIMI*, volume 90, 2003.

[86] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *International Conference on Discovery Science*, pages 16–31. Springer, 2004.

[87] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, volume 126, 2004.

[88] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 77–86. ACM, 2005.

[89] M. Wild. Computations with finite closure systems and implications. In *International Computing and Combinatorics Conference*, pages 111–120. Springer, 1995.

[90] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Springer, 1982.

[91] S. Yacout. Fault detection and diagnosis for condition based maintenance using the Logical Analysis of Data. In *Computers and Industrial Engineering (CIE), 2010 40th International Conference on*, pages 1–6. IEEE, 2010.

[92] M. J. Zaki. *Closed Itemset Mining and Non-redundant Association Rule Mining*, pages 365–368. Springer US, 2009.

[93] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 326–335. ACM, 2003.