Jan Outrata

Computing and Applying Formal Concepts

Algorithms and Methods

~ Habilitation Thesis ~

Olomouc, 2015

Address of the author

Jan Outrata
Department of Computer Science
Faculty of Science
Palacký University Olomouc
17. listopadu 12
CZ-771 46 Olomouc
Czech Republic
email: jan.outrata@upol.cz
web: outrata.inf.upol.cz

Keywords: formal concept analysis; concept lattices; data mining; data analysis; algorithms; classification; decision trees; feature extraction; data preprocessing; attribute sorting; Boolean matrix factorization

Věnuji své babičce († 1. červen 2014), in memoriam

Preface

Data mining and knowledge discovery are buzzwords for data analysis and processing in both theoretical and applied computer science today. New methods and algorithms are sought to help cope with the ever-growing amount of data produced by modern society. This thesis contributes to these fields by cultivating a particular data mining and data analysis method, called *Formal Concept Analysis (FCA)*. FCA is a modern and intensively studied method for mining and analysis of object-attribute relational data with strong mathematical foundations. Since its start in the 1980s it enjoys an increasing interest and becomes increasingly popular in more and more scientific communities of mathematicians, computer scientists, engineers, and experts from various fields. During its development, FCA has been applied in many different fields of both computer and non-computer science areas.

The thesis summarizes and further comments on selected results of research in the algorithms and applications of FCA conducted by the author at the Department of Computer Science, Palacký University Olomouc, during the years 2007–2012, with remarks to further results from years 2013–2014. In the first years, the research was focused on the topic of applying FCA in classification of data with the aim at developing a decision tree induction method based on FCA. Then, due to a growing need that appeared in several related problem areas, the focus moved to the development of efficient algorithms for computing formal concepts–the basic units of data studied in FCA–which could be effectively used in applications of FCA, most eminently in data mining. In the last years and present the focus of the research has been on applying FCA for preprocessing of data for other data mining and machine learning methods. The aim was to use Boolean matrix factorization, performed via the structures of FCA, as a method for solving the feature extraction problem.
The thesis is composed of a compact introduction to FCA and a collection of papers with commented summaries of results. The collection consists of 4 impacted journal papers and 6 peer-reviewed papers published in proceedings of international conferences. The contribution of the author of this thesis in all of the papers is at least proportional to the number of (co-)authors,

in 4 papers more than proportional and of 3 papers he is the only author. The summaries, for the sake of consistency and self-containedness and also to better show the relationships between the topics, contain also (shortened) descriptions of the algorithms and methods developed in the respective papers. This includes also pseudocodes, illustrative examples, and sample results from experimental evaluations.

Acknowledgments

I would like to thank my colleagues from the Department of Computer Science, Faculty of Science, Palacký University Olomouc, for their valuable collaboration in the presented joint research. Namely Radim Bělohlávek, Vilém Vychodil and Petr Krajča. Special thanks go to Radim Bělohlávek for being my advisor and teacher even after supervising my PhD study and for valuable comments to the pre-final version of the thesis text. Thanks go also to my friends and to my family for their patience and moral support.

Jan Outrata, author
Olomouc, June 2015

Contents

Chapter 1

Introduction

A huge amount of knowledge is stored in the currently available data. Typically, the data is hardly usable "as it is". Therefore people try to find ways to discover and extract the essential and most interesting bits of the knowledge hidden in the data, in forms of relatively small and well-interpretable structures or *patterns* which would be easier to use. This is a challenging goal. Computer methods for retrieving such structures are commonly referred to as methods of data mining or knowledge discovery, and methods for further transforming and analysis of the structures for further use then as methods of data analysis. The intended purpose of the structures is to represent well the extracted knowledge for their usage by people. Since people reason about things naturally in terms of *concepts*, a challenging goal is to find an operational abstract notion of concept that naturally "exists in" or is supported by the data. A formalization of concept formation is a basic issue in several areas of science, e.g. psychology, sociology, cognitive sciences, etc. The approaches to concept formation vary. The objects, which in the end fall under a concept, can be grouped based on a defined similarity between them. Such approach is utilized by classical clustering techniques [44, 96]. A different approach, used beside other methods by formal concept analysis, is to define concepts by means of sharing attributes.

Formal Concept Analysis (shortly FCA), as a method of data mining and data analysis, has been initiated by Rudolf Wille at TU Darmstadt in the early 1980s, as part of his program of restructuring lattice theory [136]. The method is based on the interpretation of concepts inspired by a traditional understanding of concept which goes back to traditional Port-Royal logic [38, 67, 68]. According to Port-Royal, a concept has two parts—its *extent* and *intent*. The extent is a collection of objects which fall under the concept while the intent is a collection of attributes covered by the concept. For instance, the extent of the concept "bird" is the collection of all birds (as objects) while its intent consists of all properties (as attributes) of birds like "flies", "has feathers", etc. FCA formalizes the notion of concept by

the notion of a *formal concept*. A formal concept, in terms of FCA, is an (ordered) pair of two collections (sets)—the collection of objects and the collection of attributes, having the crucial (defining) property that the collection of objects is the collection of all objects sharing all attributes from the collection of attributes and, conversely, the collection of attributes is the collection of all attributes shared by all objects from the collection of objects.

Formal concepts introduced above are extracted from data which describe objects by their attributes, i.e. form the so-called *object-attribute relational data*. The collection of all formal concepts extracted from the data is the basic output of FCA. The data comes usually in tabular form of a two-dimensional data table in which rows correspond to objects and columns correspond to attributes. Note that such form of data is a fundamental one in data mining and data analysis and is also the basic one for relational databases. In basic setting of FCA, a table entry for a particular object and a particular attribute indicates the presence/absence of the attribute for the object, i.e. the fact that the object "has" the attribute in the relation of "having" between objects and attributes. The presence is usually denoted by $\times$ (cross) or 1 in the table entry, the absence by an empty entry or 0. An illustration of an input data table for FCA is depicted in Figure 1.1 (left). Because of the two possible "values" of an attribute for each object, the attributes are termed bivalent or binary attributes. More general attributes, like categorical (nominal), ordinal, or numerical, are in FCA handled by so-called *conceptual scaling*. This provides a particular transformation of a data table with general attributes to a data table with binary attributes which, in a certain sense, respects the original meaning of the attributes. We refer to [51] for details. For graded (or fuzzy, as commonly called) attributes several generalizations of (the ordinary) FCA to FCA with graded (fuzzy) attributes (sometimes misleadingly called "fuzzy FCA") have been proposed [11, 17, 21, 23, 33, 78, 117, 139], see [12, 30] for an overview. However, the most appealing seems to be the approach proposed independently by Pollandt [117] and Belohlavek [9] which uses residuated scales of grades [53, 58].

Some concepts are more general than others in that they apply to more objects and cover less attributes. For instance, the concept "mammal" is more general than the concept "dog" which is more specific. Similarly for the concepts "dog" and "labrador retriever". This subconcept-superconcept hierarchy, studied already by Port-Royal, represents the specificity-generality relationship and plays a fundamental role in FCA. The collection of all formal concepts extracted from data together with the subconcept-superconcept hierarchy is called the *concept lattice* of the data. A concept lattice is the basic output of FCA for its applications. Concept lattices are usually depicted by labeled line diagrams (Hasse diagrams). For illustration, the concept lattice of the data table from Figure 1.1 (left) is depicted in Figure 1.1 (right).

I	0	1	2	3	4	5	6	7
a		×	×		×	×		×
b	×			×			×	×
c	×	×	×	×		×	×	
d		×			×	×		
e	×	×			×	×	×	×

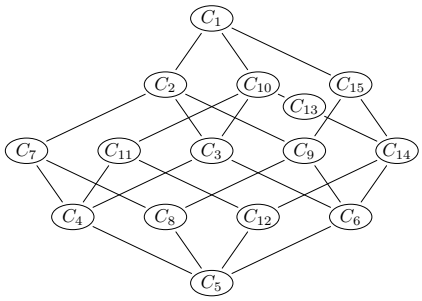

Figure 1.1: Object-attribute data table (left) and corresponding concept lattice (right).

Note that the other output of FCA derived from the data is a non-redundant base of particular attribute dependencies called attribute implications. The implications are, however, not considered in this thesis hence not further discussed.

FCA has been, right since its start in 1980s, applied in various fields. of computer and non-computer sciences. Let us name just a few most-known applications: classification [34, 47, 82, 92], database and information (catalog) systems [18, 37, 39, 42, 46, 97, 127], information retrieval (including web retrieval) [35, 66, 69, 86, 119], software engineering [2, 41, 52, 124, 125, 131] or psychology [25, 26] and sociology [22, 24, 140]. [35, 116] provide further references and surveys. Important source of applications comes also from within the data mining area itself (where attributes are usually called items and intents of formal concepts are known as closed itemsets). Formal concepts here appear as particular rectangular patterns in the data table, see Section 1.1, where they play a crucial role and have a clear, direct interpretation. In several data mining disciplines, FCA is used for *preprocessing* the data, where the extracted formal concepts are not used directly (by users), instead they are used as input for other data mining methods. For instance, in [141] it has been shown that formal concepts can be used to find (non-redundant) association rules [1, 91, 113, 134] (intents of formal concepts with additional constraint on the number of objects are identified with the so-called frequent closed itemsets) or, by [29], formal concepts can be used to find good sub-optimal solutions for Boolean Matrix Factorization [65] and Discrete Basis [95] problems (formal concepts represent factors).

Summed up, formal concept analysis, or the theory of concept lattices, as it is sometimes referred to, is a well established and elaborated data mining and data analysis method nowadays. Results on FCA and concept lattices are being reported in premier conferences and journals on data mining and data analysis. Its strong theoretical foundations, developed at TU Darmsdadt and TU Dresden during the 1980s and start of 1990s, are summarized in [51] which is a basic (and extensive) source for FCA. Some algorithms for

computing formal concepts, concept lattices and a base of attribute implications from input data table, as well as a survey of applications, particularly in information retrieval area, can be found in [35], another well-know book on FCA and concept lattices. A good starting point to obtain information on FCA is the "FCA Homepage" web page [118]. A brief formal introduction to FCA theory is given in Section 1.1.

Outline of the thesis

The thesis consists of two main parts, commenting upon and summarizing the research of the author on computing formal concepts and two applications of formal concepts.

The task to compute the collection of all formal concepts in the input object-attribute relational data is a basic task in FCA which appears in virtually every of its applications. Extracting formal concepts from the data is therefore a crucial problem. In the history of FCA, and even before, quite many algorithms for this task have been developed. The *NextClosure* algorithm [51] is one of the simplest and is being used in comparisons as well as in introductory texts. The existing algorithms have recently became subject of criticism because they were basically developed for middle-size data (thousands of objects and about a hundred of attributes). With growing size of data, the performance is not satisfactory. This is often caused by non-efficient existing implementations of the algorithms, usually as proof-of-correctness version by their author(s) only. Another challenging problem in FCA is how to deal with large-scale data (nowadays often termed "big data"). The problem has become more important as FCA is becoming increasingly popular in the data mining community.

Our results in **algorithms for computing formal concepts** are the content of Chapter 2. We start with the base algorithm upon which all further described algorithms are based. Then we describe a parallel version of it and an enhancement which significantly improves the performance of the base algorithm. Then we show that the performance can be further increased by preprocessing the input data prior to actual computation of formal concepts. Finally, extending this idea, we come up with an algorithm with completely novel approach which further reduces the amount of redundant formal concept computation. Last but not least, we pay attention to implementation issues (regarding data representation) which are so scarcely discussed in the existing papers but have a considerable final impact on the real performance of the algorithms.

The second part of the thesis, Chapter 3, is devoted to **applying formal concepts**. We present two applications. The first one is in the the area of classification of data. The usefulness of using FCA (and concept lattices) in classification has been recognized by several authors [45, 47, 56, 82], under

the names of lattice-based or concept-based learning. FCA is used to create various classification models or to preprocess input data for classification models. We present a novel *decision tree induction* method which is based on a straightforward idea of utilizing formal concepts of input data as nodes of the decision tree constructed from the data. In contrast to other approaches in the literature based on this idea, in our approach we utilize the closure properties of formal concepts and the subconcept-superconcept hierarchy of the concepts directly in the process of construction of the decision tree, as opposed to as a preprocessing step or a basis for a new machine learning method. To compute the formal concepts the (modified) algorithms from the first part of the thesis can be used. An experimental evaluation indicates good classification performance, in comparison to standard decision tree induction and other classification methods.

The second presented application of formal concepts is in preprocessing of data before the data is input to other data analysis technique. Usage of FCA as a data preprocessing technique is often proposed in the literature [97, 133]. We present a novel method to utilize formal concepts for *feature extraction* (creating new attributes). The method is based on the recently proposed Boolean matrix factorization (BMF) method based on utilizing formal concepts as factors [29]. The factors constitute new attributes. Such usage of FCA has not yet been reported in the literature. The usefulness of this method is demonstrated and evaluated again in the classification problem. We demonstrate that the preprocessed data are better classified than the original data. To compute the formal concepts which are used to create new attributes (factors) we use a BMF algorithm that takes advantages (regarding performance) of the algorithms described in the first part of the thesis.

List of papers

The summary descriptions of the algorithms and methods in Chapters 2 and 3 are based on the following papers (sorted by topic and within a topic chronologically). For each paper in the list, numbers of citations (without self-citations by any of co-authors) as of July 2015 and a contribution of the author of this thesis to the paper are specified. In all cases the author contributed to all activities during papers preparation. The papers are cited in appropriate places in Chapters 2 and 3.

[73] p. 93 Krajca P., Outrata J., Vychodil V.: Parallel Recursive Algorithm for FCA. In: Belohlavek R., Kuznetsov S. O. (Eds.): *CLA 2008: Proceedings of the Sixth Int. Conf. on Concept Lattices and Their Applications*, 71–82, Olomouc, Czech Rep., 10/2008. CEUR WS, Vol. 433, indexed by Scopus
Citations (without self-citations): 1 Scopus, 40 total
The author's contribution: 40 % – idea, algorithm, implementation, writing.
Base for Sections 2.2.1 and 2.2.2.

[74] p. 105 Krajca P., Outrata J., Vychodil V.: Parallel Algorithm for Computing Fixpoints of Galois Connections. **Annals of Mathematics and Artificial Intelligence** 59(2)(2010), 257–272. DOI 10.1007/s10472-010-9199-5
IF: 0.430, citations (without self-citations): 3 WoS, 7 Scopus, 13 total
Full version of paper [73].
The author's contribution: 40 % – idea, algorithm, implementation, writing.
Base for Sections 2.2.1 and 2.2.2.

[71] p. 121 Krajca P., Outrata J., Vychodil V.: Advances in algorithms based on CbO. In: Kryszkiewicz M., Obiedkov S. (Eds.): *CLA 2010: Proceedings of the 7th Int. Conf. on Concept Lattices and Their Applications*, 325–337, Sevilla, Spain, 10/2010. CEUR WS, Vol. 672, indexed by Scopus
Citations (without self-citations): 21 total
The author's contribution: 33 % – algorithms, ideas, implementations, experiments, writing.
Base for Sections 2.2.3 and 2.3.2.

[111] p. 135 Outrata J., Vychodil V.: Fast Algorithm for Computing Fixpoints of Galois Connections Induced by Object-Attribute Relational Data. **Information Sciences** 185(1)(2012), 114–127. DOI 10.1016/j.ins.2011.09.023
IF: 3.643, citations (without self-citations): 7 WoS, 10 Scopus, 14 total
Full version of part of paper [71].
The author's contribution: 50 % – idea, algorithm, implementation, experiments, writing.
Base for Sections 2.2.1 and 2.2.3.

[72] p. 149 Krajca P., Outrata J., Vychodil V.: Computing formal concepts by attribute sorting. **Fundamenta Informaticae** 115(4)(2012), 395–417. DOI 10.3233/FI-2012-661
IF: 0.399, citations (without self-citations): 2 WoS, 2 Scopus, 2 total
The author's contribution: 33 % – idea, algorithm, implementation, writing.
Base for Sections 2.3.2 and 2.4.

[13] p. 173 Bělohlávek R., De Baets B., Outrata J., Vychodil V.: Inducing decision trees via concept lattices. In: Diatta J., Eklund P., Liquire M. (Eds.): *Proc. CLA 2007*, 274–285, Montpellier, France, 10/2007. CEUR WS, Vol. 331, indexed by Scopus
Citations (without self-citations): 1 total
The author's contribution: 75 % – idea, algorithm, implementation, experiments, most of writing.
Base for Section 3.1.

[109] p. 185 Outrata J.: Inducing decision trees via concept lattices. In: Trappl R. (Ed.): *Cybernetics and Systems 2008: Proceedings of the 19th European Meeting on Cybernetics and Systems Research*, 9–14, Vienna, Austria, 3/2008.
Citations (without self-citations): 0 total
The author's contribution: 100 %.
Base for Section 3.1.

[14] p. 191 Belohlavek R., De Baets B., Outrata J., Vychodil V.: Inducing decision trees via concept lattices. **Int. Journal of General Systems** 38(4)(2009), 455–467. DOI 10.1080/03081070902857563
IF: 0.611, citations (without self-citations): 9 WoS, 14 Scopus, 19 total
Full version of paper [13].
The author's contribution: 65 % – idea, algorithm, implementation, experiments, most of writing.
Base for Section 3.1.

[110] p. 205 Outrata J.: Preprocessing input data for machine learning by FCA. In: Kryszkiewicz M., Obiedkov S. (Eds.): *CLA 2010: Proceedings of the 7th Int. Conf. on Concept Lattices and Their Applications*, 187–198, Sevilla, Spain, 10/2010. CEUR WS, Vol. 672, indexed by Scopus
Citations (without self-citations): 4 total
The author's contribution: 100 %.
Base for Section 3.2.

[108] p. 217 Outrata J.: Boolean factor analysis for data preprocessing in machine learning. In: Draghici S., Khoshgoftaar T. M., Palade V., Pedrycz V., Wani M. A., Zhu X. (Eds.): *Proceedings of The Ninth Int. Conf. on Machine Learning and Applications (ICMLA 2010)*, 899–902, Washington, D.C., USA, 12/2010. DOI 10.1109/ICMLA.2010.141, indexed by Scopus, included in CORE Conference Ranking (rank C)
Citations (without self-citations): 7 Scopus, 11 total
The author's contribution: 100 %.
Base for Section 3.2.

Other selected papers of the author

The following are further selected papers of the author related to topics of the thesis. A short characteristics is given for both papers.

Belohlavek R., De Baets B., Outrata J., Vychodil V.: Computing the lattice of all fixpoints of a fuzzy closure operator. **IEEE Trans. Fuzzy Systems** 18(3)(2010), 546–557. DOI 10.1109/TFUZZ.2010.2041006
IF: 2.695, citations (without self-citations): 19 WoS, 31 Scopus, 34 total

– An extension of the Lindig's *NextNeighbor/UpperNeighbor* algorithm [87] to compute the lattice of all fixpoints of a closure operator, in particular the concept lattice of object-attribute data, from the setting of Boolean attributes to graded (fuzzy) attributes [9, 10].

Bělohlávek R, Dvořák J., Outrata J.: Fast factorization by similarity in formal concept analysis of data with fuzzy attributes. **Journal of Computer and System Sciences** 73(6)(2007), 1012–1022. DOI 10.1016/j.jcss.2007.03.016
IF: 1.185, citations (without self-citations): 9 WoS, 14 Scopus, 22 total

– Fuzzy concept lattice factorization by similarity relation directly from input data with graded (fuzzy) attributes (without the need to compute the whole concept lattice) in the approach of FCA with graded (fuzzy) attributes [9, 10].

1.1 Preliminaries in formal concept analysis

Before going to the main parts of the thesis, let us first summarize formally the basic notions of formal concept analysis (FCA) which were used informally in the introduction. We assume here basic knowledge of some notions from algebra (binary relations, partial orders, complete lattices, Hasse diagrams).

An *object-attribute data table* can be identified with a triplet $\langle X, Y, I\rangle$ where X is a (finite) non-empty set of objects, Y is a (finite) non-empty set of attributes, and $I \subseteq X \times Y$ is a binary relation between set X of objects and set Y of attributes. In the relation, $\langle x, y\rangle \in I$ indicates that object x has attribute y. In the table, used to visualize the relation, objects correspond to table rows, attributes correspond to table columns, and if $\langle x, y\rangle \in I$ then the table entry corresponding to row x and column y contains (usually) $\times$ or 1, otherwise it contains blank symbol or 0. In terms of FCA, $\langle X, Y, I\rangle$ is called a *formal context*. Now, for every $A \subseteq X$ and $B \subseteq Y$ denote by $A^{\uparrow_I}$ a subset of Y and by $B^{\downarrow_I}$ a subset of X defined as

$$A^{\uparrow_I} = \{y \in Y \mid \text{for each } x \in A:\ \langle x, y\rangle \in I\}, \tag{1.1}$$

$$B^{\downarrow_I} = \{x \in X \mid \text{for each } y \in B:\ \langle x, y\rangle \in I\}. \tag{1.2}$$

That is, $A^{\uparrow_I}$ is the set of all attributes from Y shared by all objects from A, and $B^{\downarrow_I}$ is the set of all objects from X sharing all attributes from B. Operators $^{\uparrow_I} : 2^X \to 2^Y$ and $^{\downarrow_I} : 2^Y \to 2^X$ defined by (1.1) and (1.2) are called *concept-forming operators* induced by formal context $\langle X, Y, I\rangle$. If there is no danger of confusion, we usually omit I and write just $^{\uparrow}$ and $^{\downarrow}$ instead of $^{\uparrow_I}$ and $^{\downarrow_I}$. Note that the operators form a so-called Galois connection [55, 105, 136] induced by the binary relation I and the compound operators $^{\uparrow\downarrow}$ and $^{\downarrow\uparrow}$ are induced closure operators in X and Y, respectivelly. Any couple $\langle A, B\rangle \in 2^X \times 2^Y$ such that $A^{\uparrow} = B$ and $B^{\downarrow} = A$ is then called a *formal concept in* $\langle X, Y, I\rangle$. Thus, formal concepts are (so-called) fixed points of the concept-forming operators induced by $\langle X, Y, I\rangle$.
Formal concepts represent basic structures (patterns) that can be found in object-attribute data tables and have basically two interpretations: (i) a conceptual one, discussed already in the introduction, where each formal concept $\langle A, B\rangle$ represents a concept in data with an extent A, as objects that fall under the concept, and an intent B, as attributes covered by the concept, such that A is a set of all objects sharing all attributes from B and B is the set of all attributes shared by all objects from A—the interpretation inspired by a traditional understanding of concept going back to traditional Port-Royal logic [38, 67, 68]; (ii) a geometric one, where, informally, formal concepts correspond to maximal rectangular areas in the data table (*maximal rectangles*) full of $\times$s (or 1s). The geometric interpretation

is important from the point of view of data mining, as mentioned in the introduction. Note also that due to being fixed points of Galois connections (and closure operators), formal concepts are also important from the mathematical point of view (the mathematics of formal concepts is indeed essential for algorithms for generating the concepts).
Furthermore, according to Port Royal, one may consider the *subconcept-superconcept hierarchy* on formal concepts. Formally, the set $\mathcal{B}(X,Y,I) = \{\langle A,B\rangle \mid A^{\uparrow} = B, B^{\downarrow} = A\}$ of all formal concepts in $\langle X,Y,I\rangle$ can be equipped with a partial order $\leq$ defined by

$$\langle A_1,B_1\rangle \leq \langle A_2,B_2\rangle \text{ iff } A_1 \subseteq A_2 \text{ (iff } B_2 \subseteq B_1). \tag{1.3}$$

The partial order models a subconcept-superconcept hierarchy. The set $\mathcal{B}(X,Y,I)$ equipped with $\leq$, denoted $\langle \mathcal{B}(X,Y,I), \leq\rangle$, happens to be a complete lattice, called the *concept lattice of* $\langle X,Y,I\rangle$. The basic structure of concept lattices is described by the so-called Basic or Main theorem of concept lattices [51]:

Theorem 1 (Main theorem of concept lattices [51])
(1) *The set $\mathcal{B}(X,Y,I)$ equipped with $\leq$ forms a complete lattice in which infima and suprema are given by*

$$\begin{aligned} \textstyle\bigwedge_{j\in J}\langle A_j,B_j\rangle &= \langle \textstyle\bigcap_{j\in J} A_j, (\bigcup_{j\in J} B_j)^{\downarrow\uparrow}\rangle, \\ \textstyle\bigvee_{j\in J}\langle A_j,B_j\rangle &= \langle (\textstyle\bigcup_{j\in J} A_j)^{\uparrow\downarrow}, \bigcap_{j\in J} B_j\rangle. \end{aligned}$$

(2) *Moreover, an arbitrary complete lattice $\mathbf{V} = \langle V,\leq\rangle$ is isomorphic to $\langle \mathcal{B}(X,Y,I),\leq\rangle$ iff there are mappings $\gamma : X \to V$, $\mu : Y \to V$ such that*

(i) *$\gamma(X)$ is $\bigvee$-dense in V, $\mu(Y)$ is $\bigwedge$-dense in V, and*

(ii) *$\gamma(x) \leq \mu(y)$ iff $\langle x,y\rangle \in I$.*

Recall that a subset $K \subseteq V$ is $\bigvee$-dense in V ($\bigwedge$-dense in V) if for every $v \in V$ there is $K' \subseteq K$ such that $v = \bigvee K'$ ($v = \bigwedge K'$).
Note that, as a complete lattice, a concept lattice can be depicted by means of a labelled Hasse diagram which represents the cover relation on $\mathcal{B}(X,Y,I)$ (the cover relation on $\mathcal{B}(X,Y,I)$ is defined as follows: a formal concept $\langle A_1,B_1\rangle$ covers a formal concept $\langle A_2,B_2\rangle$ if $\langle A_2,B_2\rangle \leq \langle A_1,B_1\rangle$ and there is no $\langle A_3,B_3\rangle$ distinct from both $\langle A_1,B_1\rangle$ and $\langle A_2,B_2\rangle$ such that $\langle A_2,B_2\rangle < \langle A_3,B_3\rangle < \langle A_1,B_1\rangle$).
For more information on theoretical foundations, methods and algorithms of formal concept analysis and its applications in various areas we refer the reader to [35, 50, 51] and conclude this section with a small illustrative example.

I	0	1	2	3	4	5	6	7
a		×	×		×	×		×
b	×			×			×	×
c	×	×	×	×		×	×	
d		×			×	×		
e	×	×			×	×	×	×

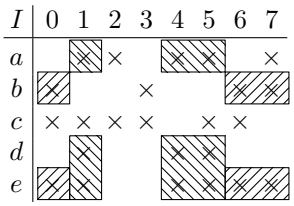

Figure 1.2: Formal context (left) and maximal rectangles (right) corresponding to formal concepts C_9 and C_{13}.

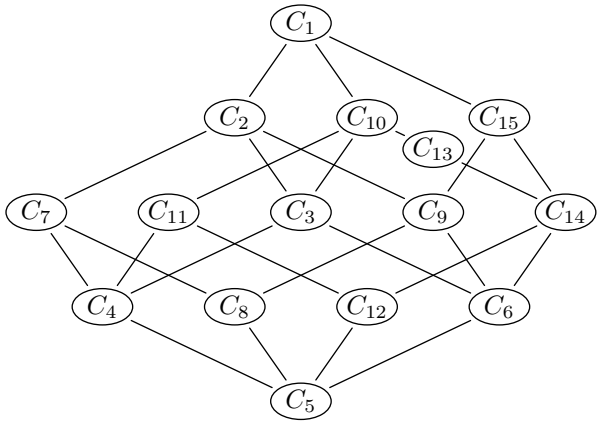

Figure 1.3: Concept lattice of formal context from Figure 1.2.

Example 1 *Consider the formal context $\langle X, Y, I\rangle$ corresponding to the object-attribute data table depicted in Figure 1.2 (left). The concept-forming operators induced by this formal context have exactly* 15 *fixed points (formal concepts) $C_1, \ldots, C_{15}$:*

$C_1 = \langle X, \emptyset\rangle$,
$C_2 = \langle\{b, c, e\}, \{0, 6\}\rangle$,
$C_3 = \langle\{c, e\}, \{0, 1, 5, 6\}\rangle$,
$C_4 = \langle\{c\}, \{0, 1, 2, 3, 5, 6\}\rangle$,
$C_5 = \langle\emptyset, Y\rangle$,
$C_6 = \langle\{e\}, \{0, 1, 4, 5, 6, 7\}\rangle$,
$C_7 = \langle\{b, c\}, \{0, 3, 6\}\rangle$,
$C_8 = \langle\{b\}, \{0, 3, 6, 7\}\rangle$,
$C_9 = \langle\{b, e\}, \{0, 6, 7\}\rangle$,
$C_{10} = \langle\{a, c, d, e\}, \{1, 5\}\rangle$,
$C_{11} = \langle\{a, c\}, \{1, 2, 5\}\rangle$,
$C_{12} = \langle\{a\}, \{1, 2, 4, 5, 7\}\rangle$,
$C_{13} = \langle\{a, d, e\}, \{1, 4, 5\}\rangle$,
$C_{14} = \langle\{a, e\}, \{1, 4, 5, 7\}\rangle$,
$C_{15} = \langle\{a, b, e\}, \{7\}\rangle$.

For illustration, the interpretation of formal concepts C_9 and C_{13} as maximal rectangles is depicted in Figure 1.2 (right). If we equip $\mathcal{B}(X, Y, I) = \{C_1, \ldots, C_{15}\}$ with the partial order $\leq$ (1.3), the resulting structure is the concept lattice $\langle\mathcal{B}(X, Y, I), \leq\rangle$ of $\langle X, Y, I\rangle$. Hasse diagram of the lattice is depicted in Figure 1.3.

Chapter 2

Computing formal concepts

2.1 Introduction and state-of-the-art

The first main part of the thesis is devoted to the problem of computing all formal concepts from a given object-attribute data table.

Quite many algorithms for computing formal concepts were developed in the history of FCA, with quite varying, but *polynomial time delay* [63]. Practically, the time (and memory) efficiency of the algorithms is varying even more, see [84] and [126] for overviews and comparisons. From the point of view of maximal rectangles, all the algorithms search for formal concepts as maximal rectangles in the space of all possible rectangles computing and listing just the maximal ones. The main difference among the algorithms is in particular ways in which they traverse through the search space. In a broader sense, the algorithms for computing formal concepts belong to an important family of algorithms for listing combinatorial structures [54] and algorithms for biclustering [5, 96].
An important issue solved by all algorithms for computing formal concepts is that in the search some formal concepts are computed multiple times while each formal concept has to be processed (e.g. stored or listed) exactly once. The issue can be solved by designing the algorithm in such a way that either computing any formal concept more than once is disallowed, or it is avoided to list the same formal concept more that once. The former approach, although more appealing, is rather difficult to achieve with a reasonable overhead because of maintaining complex data structures needed for that. The approach is used for instance by Berry's algorithm [31]. The latter approach, more favorable and most often used in existing algorithms, usually lies in a test ensuring that any formal concept, if computed multiple times, is listed only once. The test can be realized in several different ways. For instance, Lindig's algorithm [87], also named as *NextNeighbor* or *UpperNeighbor* in the literature, stores and looks for the presence of computed formal concepts in an additional data structure (typically a search tree or a

hash table). However, as with the previous approach, maintaining and, more importantly, searching the data structure renders the algorithm not very efficient, even for data with hundreds of objects and hundred of attributes. The great advantage of the algorithm is computing also the concept hierarchy in addition to all formal concepts, hence the concept lattice of the data. Here lies the primary usage of the algorithm, mainly in applications of FCA. On the other hand, the algorithm proposed by Norris [103, 104], one of the first proposed for the task, *NextClosure* algorithm [49, 51], also known as Ganter's, Kuznetsov's *Close-by-One* algorithm [79, 80, 81], and many other algorithms best known in the literature use a so-called *canonicity test* to ensure that formal concepts are listed in a unique, predefined, order. If a newly, possibly partly, computed formal concept does not pass the canonicity test, because it was computed "out of the order", it is not listed. Hence, the canonicity test ensures that even if a formal concept is computed several times, it is listed exactly once. Although conceptually similar, each algorithm uses different particular form of the test which influences the real efficiency of the algorithm. For instance, while *NextClosure* lists all formal concepts in a lexical order, which is more expensive to enforce, *Close-by-One*, although based on a very similar idea, uses more efficient order and test and runs much faster. Recently, an increasing attention has been paid to *Close-by-One*. One of the most recent and most efficient modifications of it is the *InClose* algorithm [3] and its improved version [4] proposed by Andrews.

The algorithms described in the following sections can also be conceptually seen as variants of the *Close-by-One (CbO)* algorithm. Actually, they are derivatives of a recursive version of *Close-by-One*, and since they use the same (or improved) canonicity test for avoiding to list the same formal concept multiple times, we call them CbO-family algorithms [72]. The recursive base version of *CbO*, upon which all further described algorithms are based, was first presented in [135] (which actually "rediscovered" the original *Close-by-One*) and is briefly summarized in Section 2.2.1 and in details presented also in [73, 74, 111].
Now, almost all algorithms proposed in the literature to date, are sequential or serial ones. However, with the recent increasing interest in parallel computing and growing affordability of multicore processors and other hardware allowing parallel computations these days, parallel algorithms are preferred and, actually, required, to better utilize the hardware. Therefore, in Section 2.2.2, we summarize a parallel version of (the recursive version of) *CbO*, called *Parallel CbO (PCbO)*, which can be run in multiple independent processes on multiple processor cores or processors. We show a clear and efficient way to parallelize the computation of formal concepts by splitting the set of all formal concepts into disjoint subsets which can be computed simultaneously with virtually no overhead. Indeed, the distinctive feature

of the approach compared to other existing approaches to parallel version of known algorithms, which has positive impacts on its performance and scalability, is that it completely avoids any but output synchronization. A more detailed description of the approach and the algorithm than in Section 2.2.2 can be found in [73, 74].

Next, having algorithms which compute the same formal concept more than once, the common drawback of the algorithms is that the total number of computed formal concepts is usually much (several times) bigger than the number of listed formal concepts for both real and artificial data. That is, there are several times more repeated computations of formal concepts which are not listed because of the failed canonicity test than the number of formal concepts which are listed. This indeed has a (negative) impact on the performance of the algorithms because the computation of a single formal concept is the most critical operation in any algorithm. So the aim is to design an algorithm in such a way that the total number of computed formal concepts, or the number of formal concepts computed multiple times, is as low as possible. We modified, and actually improved in this respect, Kuznetsov's *CbO* algorithm, so that we introduced a more efficient canonicity test which significantly reduces (though not completely eliminates) the number of formal concepts which are computed multiple times. Due to the performance improvement coming from the reduction we call the improved algorithm *Fast CbO (FCbO)*. A summarization of it is a subject of Section 2.2.3, with a more detailed description in [71, 111].

Since the presented algorithms compute some formal concepts multiple times and, as mentioned above, the computation of a single formal concept is crucial for the algorithms, it is very important to have a computation of a single formal concept as efficient as possible. Procedure presented in Section 2.2.4 has an advantage over conventional procedures (which directly implement the concept-forming operators (1.1) and (1.2)) in that it goes over the input data table only once instead of twice, taking advantage of the properties of the concept-forming operators.

Furthermore, the number of formal concepts computed multiple times can be made much smaller, and hence the performance of the algorithm significantly higher, by *preprocessing input data* before the actual computation of formal concepts. This issue is often underestimated in literature (not only on FCA algorithms). In fact, most of the algorithms for FCA, including the algorithms based on *Close-by-One*, when designed to search for formal concepts by iterating over input data attributes, compute significantly less formal concepts multiple times and achieve thus significantly better performance if the attributes (columns of the input data table) are processed in a particular order. This is related to the canonicity test and in Section 2.3.2 we show which order suits to algorithms from the CbO-family, with more details to be found in [71].

Moreover, elaborating on the idea of the (proper) ordering of attributes a bit

more, in essence extending it to a novel approach of ordering the (remaining) attributes not present in the obtained formal concept after each concept listing, one can come up with a bright new algorithm—an algorithm for which the number of formal concepts computed multiple times reduces to a small fraction of the total number of computed formal concepts. The algorithm combines basic ideas of (the recursive version of) *Close-by-One* with the (proper) ordering of attributes and successive formal context reduction. We summarize this last, *attribute sorting* algorithm, w.r.t. the total number of computed formal concepts the most evolved among our algorithms derived from *Close-by-One*, in Section 2.4, with the description borrowed from [72].

The last but not least issue, which (really) considerably affects the real performance of any algorithm and which is, unfortunately, very scarcely discussed in the literature, is implementation. Authors of existing algorithms typically describe their algorithm and do basic comparison to referential algorithms (Ganter's *NextClosure* or Lindig's *NextNeighbor*) and do not pay any attention to implementation of the algorithm. We do. The cutting-edge performance of the algorithms presented in this thesis can be ensured by using proper data structures in their implementation. In all our implementations we use *bitwise level data structures* to represent both the input data and computed formal concepts, see Section 2.3.1 for a clarification of this choice and details. This representation allows to take advantage of low-level operations present in contemporary microprocessors (and, these days, also graphic processors, see [85, 123]) which really considerably, by order of several magnitudes, improves the actual performance of the algorithms (in particular, the procedure for computing a single formal concept, presented in Section 2.2.4).

To show that, but also the performance of the algorithms themselves disregarding a particular implementation, we include, in Section 2.5, some results from several performance evaluations we had performed on selected real datasets and a basic comparison to Ganter's *NextClosure* algorithm. More experiments with more real and artificial data, including comparisons to other algorithms for computing formal concepts known from the literature, can be found in the respective papers. In this context, let us (again) state that very surprisingly, almost all algorithms developed by FCA community and proposed in the literature, to recent time, even if implemented with efficiency in mind, are sufficient regarding performance for middle-size data only, up to thousands to tens thousand of objects and a hundred of attributes. With growing data size, the performance of the algorithms is not satisfactory (though a well-suited implementation may help a bit). Our algorithms and, in particular, the implementations of them, run in reasonable time for data with size going up to tens to hundreds thousand of objects and hundreds to thousands of attributes.

2.2 New CbO-family algorithms

We are now ready to describe the algorithms. Due to computability reasons we obviously restrict the sets of objects and attributes to be finite nonempty sets. Let $X = \{0, 1, \ldots, m\}$ be our set of objects and $Y = \{0, 1, \ldots, n\}$ be our set of attributes.
Note that in order to compute all formal concepts it is sufficient to compute all intents of the concepts because each formal concept is uniquely determined by its intent. Namely, if $\langle A, B\rangle$ is a formal concept in $\langle X, Y, I\rangle$ with extent $A \subseteq X$ and intent $B \subseteq Y$, then $A = B^{\downarrow}$. Analogously, each formal concept is also uniquely determined by its extent because $B = A^{\uparrow}$. Second, intents (and analogously extents) can be characterized by their closure properties. Namely, $\langle A, B\rangle$ is a formal concept iff $B = B^{\downarrow\uparrow}$ ($A = A^{\uparrow\downarrow}$), i.e. iff B (A) is a fixed point of the closure operator $^{\downarrow\uparrow}$ ($^{\uparrow\downarrow}$).

2.2.1 Recursive *CbO*

We start by briefly describing a recursive version of the Kuznetsov's *Close-by-One (CbO)* algorithm, upon which we base the algorithms described in the following sections. The detailed description can be found in [73, 74, 111, 135].

CbO has been introduced in [80] and [79] (a paper in Russian) and later used and described in [81]. The algorithm lists all formal concepts of a formal context by a systematic search in the space of all formal concepts, avoiding to list the same concept multiple times by performing a canonicity test. In [81], *CbO* is described in terms of backtracking with a construction of a particular tree of computed formal concepts, called *CbO-tree*, which is then used to induce the concept lattice hierarchy of computed formal concepts. However, in our version of *CbO* [135] we are not interested in the hierarchy, so we do not need and do not construct the *CbO-tree*. Also, we utilize a procedure for computing a single formal concept which results to a much better performance of the algorithm and for this the backtracking approach is not suitable. Our version of the algorithm of *CbO* is formalized by a recursive procedure performing a depth-first search in the space of all formal concepts. Hence we call it "recursive CbO". This type of description, in addition to technical benefits which improve its performance, is much closer to the actual implementation of the algorithm than the abstract description from [81]. Thus, in our opinion, it sheds more light on the algorithm.
Briefly, the procedure starts with an initial formal concept $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$ and the first attribute 0 and during the search, it, for all attributes of the input formal context not already present in intent of the current formal concept, first computes a new formal concept R by adding the attribute to intent of the current formal concept and makes a closure of the union (of the

intent and the attribute) by applying the closure procedure described in Section 2.2.4. Then, it is checked whether R has already been found during the search. If not, it processes R (e.g., prints it on the screen or stores it), and proceeds with computing further formal concepts resulting from R by adding further attributes to its intent. Here the procedure recursively calls itself with R being the current formal concept and one of the further attributes. If R has already been found, it is discarded. Hence the check implements a canonicity test.

The key issue is to have the canonicity test quick, without searching some data structure. To ensure this we compute the new formal concepts in a unique order, by adding attributes to intent of the current formal concept in a selected, but fixed, order. The order, together with the check, ensures that each formal concept is processed exactly once. The principle is the following. Let $\langle A, B\rangle$ be a formal concept, $j \in Y$ such that $j \notin B$. Put $D = (B \cup \{j\})^{\downarrow\uparrow}$, i.e. the new formal concept is $\langle (B \cup \{j\})^{\downarrow}, D\rangle$. Once D is computed, we check whether

$$B \cap Y_j = D \cap Y_j \tag{2.1}$$

where $Y_j \subseteq Y$ is defined by

$$Y_j = \{y \in Y \mid y < j\}. \tag{2.2}$$

is true. The condition represents the canonicity test. It expresses the fact that the closure D of $B \cup \{j\}$ does not contain any new attributes which are "before j" w.r.t. the order in which we add attributes. Together with adding attributes to B in this order condition (2.1) is used to check whether we should process D. If (2.1) is false, we do not process D because due to the depth-first search method, D has already been processed in some other branch of computation. Hence, the canonicity test prevents a formal concept from being listed multiple times. Then, after finitely many steps, the algorithm lists all formal concepts of an input formal context, each of them exactly once.

A pseudocode of the algorithm in the form of recursive procedure GenerateFrom is depicted in Algorithm 1. The procedure uses procedure ComputeClosure for computing a new formal concept the pseudocode of which is depicted in Section 2.2.4. See [73, 135] for a step-by-step description of procedure GenerateFrom. In order to compute all formal concepts of $\langle X, Y, I\rangle$, the procedure is to be invoked with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$ and $y = 0$ as arguments.

A proof of correctness for the original *CbO* algorithm by Kuznetsov is elaborated in [80] and [79]. Since we have the algorithm formulated as a recursive procedure rather than using backtracking, an independent proofs of its correctness are provided first in [74] and then in [111].

Algorithm 1: Procedure GenerateFrom($\langle A, B\rangle, y$)

Input: formal concept $\langle A, B\rangle$, number $y \in Y \cup \{n+1\}$ such that $y \notin B$
Uses : set Y of attributes, number n of attributes, procedure ComputeClosure

```
1  list ⟨A, B⟩ (e.g., print it on screen);
2  if B = Y or y > n then
3  |  return
4  end
5  for j from y upto n do
6  |  if j ∉ B then
7  |  |  set ⟨C, D⟩ to ComputeClosure(B, j);
8  |  |  if B ∩ Y_j = D ∩ Y_j then
9  |  |  |  GenerateFrom(⟨C, D⟩, j + 1);
10 |  |  end
11 |  end
12 end
13 return
```

The computation of Algorithm 1 can be depicted by a tree as that in Figure 2.1. The tree contains two types of nodes: (i) nodes represented by couples $\langle C_i, y_i\rangle$ corresponding to invocations of GenerateFrom with the arguments C_i (a formal concept) and y_i (an attribute), and (ii) leaf nodes denoted by black squares representing computed formal concepts for which the canonicity test fails. Edges in the tree are labeled by the values of j which are used to compute new formal concepts. Note that nodes (i) are in a one-to-one correspondence with formal concepts of $\mathcal{B}(X, Y, I)$. We call such a tree a *call tree* of GenerateFrom for given $\langle X, Y, I\rangle$ and we will need it in Sections 2.2.2 and 2.2.3 devoted to parallel and improved version of the algorithm.

From the point of view of the worst-case time complexity, the algorithm has asymptotic polynomial time delay [63] $O(|Y|^3 \cdot |X|)$ and asymptotic overall time complexity $O(|\mathcal{B}(X, Y, I)| \cdot |Y|^2 \cdot |X|)$ [80, 79, 83], which is common to many other algorithms for computing formal concepts, see [84, 126].

Comparing to other algorithms for computing formal concepts, it is important to note that the algorithm, as well as the original *CbO* itself, is conceptually equivalent to Ganter's *NextClosure* algorithm [49, 51]. It uses the same canonicity test to ensure that no formal concept is generated multiple times, only the concepts are listed in a different order. However, the algorithm can be easily modified to produce formal concepts in the *NextClosure*'s, lexical, order. See [111] for two possible modifications. Moreover, this "*CbO* way" for obtaining concepts in the lexical order, by a recursive ap-

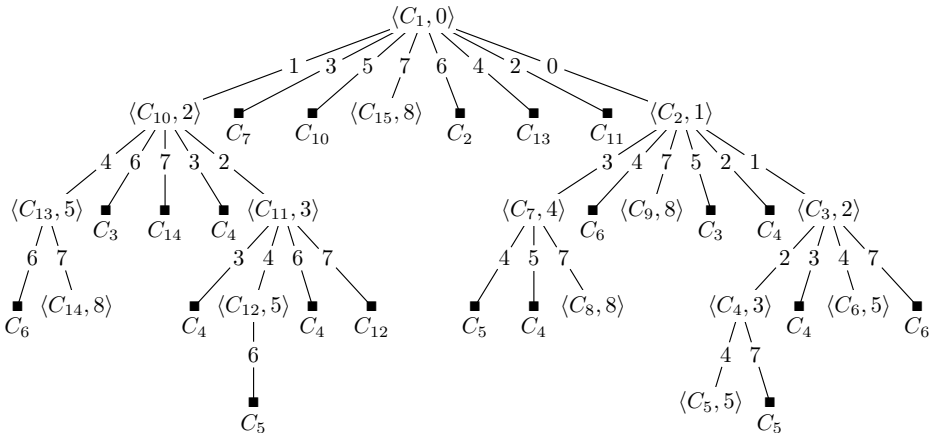

Figure 2.1: Call tree of GenerateFrom($\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle, 0$) with input data from Figure 1.2.

proach, is much faster than the iterative approach from [49, 51]. This is also, but not only, due to the more efficient computation of a single formal concept described in Section 2.2.4. The algorithm is also similar to Lindig's *NextNeighbor* algorithm [87], in that it performs a depth-first search through the search space of all formal concepts, but the key difference, which has a great impact on the performance, is the way how the algorithms test that new formal concept has already been found (recall that *NextNeighbor* stores previously computed formal concepts in an additional data structure and looks up for the new concept there). Finally, the algorithm is also related to the algorithm proposed by Norris [103, 104] which can be seen as an incremental variant of *CbO*.
Selected results from a thorough performance evaluation and comparison with some of the mentioned other algorithms are presented in Section 2.5. For more results see [135], where also a detailed listing of computation of the algorithm on example data can be found.

2.2.2 *Parallel CbO*

After we have described the base algorithm of our *CbO* variants we can introduce a parallel version of the algorithm.

Assume we can execute instructions simultaneously in multiple *independent processes*. These may be represented by operating system processes or threads (light-weight processes) running in parallel on modern multicore processors or multiple processors in a system with shared memory or on separate computers in a distributed environment within a computer network (see also the paragraph "Distributed algorithm" below on the latter). We further assume that each process has access to the input data $\langle X, Y, I \rangle$.

Since $\langle X, Y, I \rangle$ is not altered during the computation, each process can have its own copy of $\langle X, Y, I \rangle$ or processes can share one copy (in environments with shared memory).
In the following we briefly describe our approach to compute formal concepts in a given fixed number P of separate processes running in parallel. The sequential (or serial) version of the algorithm, described in the previous section as recursive *CbO*, lists all formal concepts using a depth-first search through the space of all formal concepts. The parallel algorithm can be seen as several instances of the sequential version working simultaneously on disjoint subsets of formal concepts. The parallelization consists in modifying the procedure GenerateFrom from the previous section so that particular subtrees of the call tree of the procedure are computed simultaneously in the P processes. Looking at a call tree like that in Figure 2.1 on page 20, at any level of the tree, we can see a set of nodes which are root nodes of disjoint subtrees. These subtrees may be processed independently by separate processes and that is the key idea of our approach. This suggests to modify GenerateFrom so that is goes down through the call tree only up to a certain predefined level L (counting from 1 in the root node level) and at the level it starts the computation of the remaining formal concepts descendant to those on the Lth level in parallel. The computation is done by invoking original GenerateFrom in multiple separate processes with a formal concept on the Lth level as the first argument. Therefore, the parallel procedure for computing formal concepts can be summarized by the following three consecutive stages:

Stage 1: Compute and list all formal concepts up to level L of the call tree.

Stage 2: Store the concepts in P *independent queues*.

Stage 3: Start P processes running in parallel: (i) let each of the processes take exactly one of the queues; (ii) for each formal concept in its queue let each process compute formal concepts using Algorithm 1 beginning with the concept picked from the queue.

Note that the key issue with the procedure is how to distribute formal concepts computed on the Lth level of the call tree into the P queues in Stage 2. In fact, by selecting a queue in which we put a concept we select a process by which all formal concepts descendant to the concept will be listed. The strategy of the distribution may influence the practical efficiency of the algorithm. Indeed, the optimal selection method should distribute all formal concepts to processes uniformly. This is, however, very hard to achieve since we do not know the distribution of formal concepts in the search space of all formal concepts until we actually compute them all and reveal the structure of the call tree. As a consequence, the distribution of workload may be in some cases somewhat unbalanced. In the below presented version of the algorithm, taken from [74], we use an ordinary round-robin scheme: the index

r of a selected queue is computed as $r = (N \mod P) + 1$ where N denotes the number of formal concepts stored in all queues so far. This scheme, albeit simple, turned out to be reasonably efficient for both the real-world datasets and randomly generated data. See the evaluation and comparison of this and several other schemes of the workload distribution that can be considered in [71]. Surprisingly, there are only small performance differences among the considered schemes, i.e., the round-robin scheme used in [73, 74] is adequate for the job.

Overall, the algorithm can be seen as having two parts: first, a part which distributes formal concepts into queues and, second, a part which runs several instances of the sequential (recursive) *Close-by-One* in parallel. Because of this reliance on *CbO*, we call the algorithm *Parallel Close-by-One* (*PCbO*). The algorithm is represented by procedure ParallelGenerateFrom, see Algorithm 2, a modification of the procedure GenerateFrom of the recursive *CbO* from Algorithm 1. See [73, 74] for a detailed description of the procedure, in particular the meaning of the argument l. In order to compute all formal concepts of $\langle X, Y, I \rangle$, the procedure is to be invoked with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$, $y = 0$ and $l = 1$ as its arguments.

Soundness of the algorithm follows directly from the soundness of the sequential version described in the previous section and the fact that processes compute predefined disjoint subsets of all formal concepts. Nevertheless, the complete proof of correctness of the algorithm can be found in [74].

The fact that the processes compute predefined disjoint subsets of all formal concepts also means that the processes do not interfere with each other and hence the algorithm needs no synchronization of the processes (but the synchronization of output of concepts, if applicable). The parallelization also does not increase the theoretical overall worst-case time complexity of the algorithm. The complexity remains the same as for the sequential version, (recursive) *CbO*, namely $O(|\mathcal{B}(X, Y, I)| \cdot |Y|^2 \cdot |X|)$ with polynomial time delay $O(|Y|^3 \cdot |X|)$, because in the worst case, the algorithm can degenerate into *CbO* (see [74]). The actual performance in practice compared to *CbO* is indeed influenced by the number P of processes and the workload distribution among the processes. In case of optimal workload, *PCbO* can run P times faster than *CbO*, i.e. the reciprocal P^{-1} can be seen as multiplicative constant of the running time of *CbO*. In practice, however, the multiplicative constant is a bit greater than P^{-1} because (i) formal concepts are not distributed to processes uniformly and (ii) the parallelization has certain overhead, although subtle. A hint of how *PCbO* behaves for different values of P can be seen in [73, 74] and also, very briefly, in Section 2.5.

Let us in this context also shortly comment on the role of the parameter L in the Algorithm 2, since it has an impact on the distribution of computed formal concepts to the processes and hence has an influence of the practical performance of the algorithm. According to our observations, see [74] or Section 2.5 for a brief preview, if $L = 2$, most of the formal concepts are

Algorithm 2: Procedure ParallelGenerateFrom($\langle A, B\rangle, y, l$)

Input: formal concept $\langle A, B\rangle$, number $y \in Y \cup \{n+1\}$ such that $y \notin B$, number l such that $1 \leq l \leq L$
Uses : set Y of attributes, number n of attributes, procedure ComputeClosure, level $L \geq 2$ of recursion, number $P \geq 1$ of processes, queues $queue_r, 1 \leq r \leq P$ of formal concepts, procedure GenerateFrom

```
1  if l = L then
2  |  select r ∈ {1, ..., P};
3  |  store ⟨⟨A, B⟩, y⟩ to queue_r;
4  |  return
5  end
6  list ⟨A, B⟩ (e.g., print it on screen);
7  if not (B = Y or y > n) then
8  |  for j from y upto n do
9  |  |  if j ∉ B then
10 |  |  |  set ⟨C, D⟩ to ComputeClosure(B, j);
11 |  |  |  if B ∩ Y_j = D ∩ Y_j then
12 |  |  |  |  ParallelGenerateFrom(⟨C, D⟩, j + 1, l + 1)
13 |  |  |  end
14 |  |  end
15 |  end
16 end
17 if l = 1 then
18 |  for r from 1 upto P do
19 |  |  with process r
20 |  |  |  foreach ⟨⟨C, D⟩, j⟩ ∈ queue_r do
21 |  |  |  |  GenerateFrom(⟨C, D⟩, j);
22 |  |  |  end
23 |  |  end
24 |  end
25 |  wait for all processes
26 end
27 return
```

computed in one or two processes. With increasing L, formal concepts are distributed to processes more equally. On the other hand, large values of L tend to degenerate the parallel computation. In extreme, if $L \geq |Y|+1$ then all formal concepts will be computed in the first, sequential, stage because the depth of the call tree is at most $|Y|+1$. From our experiments it seems that on average, a good trade-off value is already $L = 3$ provided that $|Y|$ is large. In such a case, almost all formal concepts are computed in parallel

and are distributed to processes nearly optimally.

Distributed algorithm Before concluding this section, let us remark the distributed variant of (the recursive) *CbO* and *Parallel CbO*. Contrary to parallel computing utilizing multiple independent processor cores or processors in a single computer system with shared memory, in distributed computing multiple *independent separate computer systems* interconnected within a computer network without a shared memory are utilized. In practice, distributed computing is used in situations where parallel computing is not affordable (cost of hardware allowing large-scale parallel computations) or sufficient (unavailability of the hardware adequate to a large size of input data). On the other hand, however, distributed computing has larger overhead of the computation management compared to parallel computing. The distributed variant of (the recursive) *CbO* (or *PCbO*) using the Google's *map-reduce framework* [40], and actually a proof-of-concept of how the framework can be used for computing formal concepts, is introduced in [77]. In essence, it is based on the same idea of splitting the set of all formal concepts into disjoint subsets which can be computed simultaneously implemented for *PCbO* above. Now, however, the disjoint subsets of formal concepts are individual tree "layers" of the call tree of procedure GenerateFrom rather than subtrees of the tree (recall Figure 2.1 on page 20). Shortly and referring to the notation from Algorithm 2 of *PCbO* above, the *map* operation computes (independently) formal concepts $\langle C, D\rangle$ on an actual layer from (respective) concepts $\langle A, B\rangle$ on the upper neighbor layer and the *reduce* operation filters the concepts $\langle C, D\rangle$ using the canonicity test. The two operations are performed repeatedly layer by layer until no further concepts are computed. Hence the strategy of generating formal concepts changes from a depth-first search in the call tree, used in (the recursive) *CbO* and *PCbO*, to a breadth-first search. See [77] for a more detailed description, including results from experiments showing the scalability of the approach.

We conclude this section with some bibliographical remarks on other existing approaches to parallel and distributed algorithms for computing formal concepts. Interestingly, we are aware of only several attempts to parallelize Ganter's *NextClosure* and one distributed version of the algorithm. Namely, [48] proposes a parallelization of *NextClosure* by decomposing the set of all formal concepts into non-overlapping subsets which are computed simultaneously. This sounds similar to our approach but for *NextClosure* the decomposition and parallelization is more complex. Another, more simple and taking advantage of the iterative way of computing formal concepts, approach to parallelize the algorithm is presented in [7]. It is based on splitting the lexicographically ordered power set 2^Y into p intervals of the same length (p indicates a number of processes) executed by independent

processes using a sequential version of the algorithm. Yet a different approach is shown in [64] where the algorithm is based on dividing input data into disjoint fragments which are then computed by independent processes. This approach, however, requires a remarkable amount of synchronization of the processes. A distributed variant of *NextClosure* is presented in [138]. Similarly to the approach of [77] presented above, it uses the map-reduce framework but, alike the previous approach, input data are first divided into disjoint fragments which are independently processed by the map operation and then the partial formal concepts are merged by the reduce operation. Also, a different implementation of the framework better supporting iterative algorithms is used.

2.2.3 *Fast CbO*

Now we turn our attention to an improvement of the canonicity test used by *CbO* that, as introduced in Section 2.1, reduces the number of formal concepts computed multiple times and, in consequence, significantly improves performance of the algorithm.

In a call tree like that in Figure 2.1 on page 20, formal concepts which are computed multiple times are depicted by the black square nodes. Our new test, and the improved algorithm with the test, reduces the number of such nodes in the call tree without altering the rest of the tree. Namely, the major "problem" with the original canonicity test used by *CbO* is that it is always used *after* a new formal concept is computed, discarding the computed concept had the canonicity test failed. As an improvement, we do not modify the original canonicity test itself. We propose an extension of the test by employing an additional test that is performed *before* a new formal concept is computed, eliminating thus the (expensive) computation of non-canonical concepts.
To explain the additional test, let us first inspect the original canonicity test. In it, for $B \subseteq Y$ and $j \notin B$, one checks whether

$$B \cap Y_j = D \cap Y_j, \text{ where } D = (B \cup \{j\})^{\downarrow\uparrow} \tag{2.3}$$

and $Y_j = \{y \in Y \mid y < j\}$, cf. 2.1 and Algorithm 1 (line 8). In words, one checks whether B and D agree on all attributes which are smaller than j (in a fixed order of attributes). Since $^{\downarrow\uparrow}$ is a closure operator and $D = (B \cup \{j\})^{\downarrow\uparrow}$, the monotony property of $^{\downarrow\uparrow}$ yields $B \subseteq D$ (thus, it is sufficient to check just the inclusion $B \cap Y_j \supseteq D \cap Y_j$ instead of the equality (2.3)). Hence, the test (2.3) fails (i.e., the equality is not true) iff $D = (B \cup \{j\})^{\downarrow\uparrow}$ contains an attribute which is "before j" and the attribute is not present in B. Let us denote all such attributes by $B \oslash j$, i.e.

$$B \oslash j = (D \setminus B) \cap Y_j = \big((B \cup \{j\})^{\downarrow\uparrow} \setminus B\big) \cap Y_j. \tag{2.4}$$

The following proposition shows that knowing that (2.3) fails for given B and $j \notin B$, we can conclude that the test will also fail for each $B' \supseteq B$ with $j \notin B'$ as long as $B \oslash j$ contains an attribute which is not in B', i.e. $B \oslash j \nsubseteq B'$:

Proposition 2 (On Test Failure Propagation [111]) *Let $B \subseteq Y$, $j \notin B$, and $B \oslash j \neq \emptyset$. Then, for each $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \nsubseteq B'$, we have $B' \oslash j \neq \emptyset$.*

Now, the proposition, proved in [111] as Lemma 2, allows us to extend the canonicity test to a *new*, two-part, *canonicity test*: first, new, part which is quick and does not require computing new formal concepts and second part, consisting in the original canonicity test with the newly computed formal concept but applied only if the first part succeeds. Indeed, according to the Proposition 2 if we know that $B \oslash j \neq \emptyset$ for some $j \notin B$ for the formal concept $\langle A, B\rangle$ then having a formal concept $\langle A', B'\rangle$, where $B' \supseteq B$, with $B \oslash j \nsubseteq B'$, we automatically know (without computing any other formal concepts) that we must not add j to B' because the subsequent original canonicity test would fail. Hence, $D' = (B' \cup \{j\})^{\downarrow\uparrow}$ of the intended new formal concept $\langle (B' \cup \{j\})^{\downarrow}, D'\rangle$ is not computed at all and the original canonicity test is not performed. Thus, in the new canonicity test, the first part of the test uses the observation of Proposition 2 while the second part is the original canonicity test.
Note that the additional test based on Proposition 2 is not always applicable and the original test is thus necessary. It is evident that we cannot apply the test on the top level (below the root) of the call tree, because $B \oslash j = \emptyset$. There are, however, situations where it cannot be applied on deeper levels as well. The situations are such that $B \oslash j \subseteq B'$. See [111] for an illustrated example of such a situation. In these cases we still have to perform the original canonicity test which involves computing $(B' \cup \{j\})^{\downarrow\uparrow}$. Nevertheless, the number of cases in which we actually perform the original canonicity test is considerably low compared to the number of failed original canonicity tests without the new test, as it can be seen from experiments in [71, 111].
For the sake of illustration of the resulting effect of the new canonicity test, consider the call tree in Figure 2.1 on page 20. If we apply the additional canonicity test based on Proposition 2, we in fact perform a particular tree pruning in which we omit some of the black square leaf nodes of the tree. The result is shown in Figure 2.2. The bold edges are those which remain in the call tree. The leaf nodes that are pruned are denoted in gray and the corresponding edges are dotted. Notice that not all back square leaf nodes are pruned. Nodes $C_2, C_4, C_6, C_7, C_{10}, C_{11}, C_{13}$ and C_{14} appear once (more) as those leaf nodes, meaning that the corresponding formal concepts are computed twice during the computation. The total number of formal concepts computed during the computation (by Algorithm 3 below) is 23—a

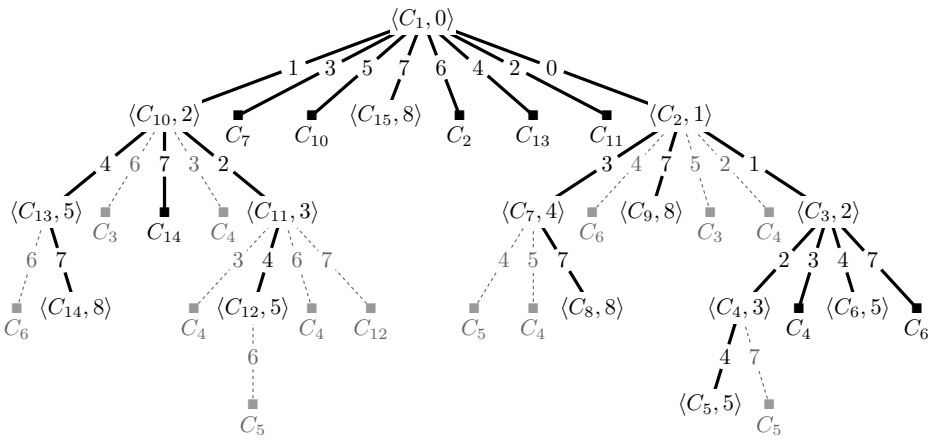

Figure 2.2: Call tree from Figure 2.1 pruned by the additional canonicity test.

significant reduction compared to 36 nodes/formal concepts of the original call tree in Figure 2.1 for the (recursive) *CbO* algorithm.

Before describing how the new canonicity test can be implemented, it is important to note that an analogous test to our additional test based on Proposition 2 appeared in the *AddIntent* algorithm introduced in [93]. *AddIntent* is, unlike *Close-by-One*, an incremental algorithm for computing all formal concepts of input data together with the subconcept-superconcept hierarchy $\leq$ given by (2.8), i.e. the concept lattice of the data. The algorithm incrementally computes the concept lattice of a given input data by adding all attributes (or objects as in [93]) one by one. The key difference between our additional test and the optimization test in *AddIntent* is that *AddIntent* uses a slightly different canonicity test that is based on the ordering $\leq$ of formal concepts, whereas our algorithms based on (recursive) *Close-by-One* use the order of processed attributes. See [111] for a description of the analogy to our additional test in *AddIntent*, using the notion of a canonical generator of a formal concept from the description of the algorithm in [93]. As a consequence, the approach of employing the test used by *AddIntent* is more beneficial if one wants to compute the whole concept lattice instead of computing just the formal concepts. On the other hand, our approach employed for *Close-by-One* and described below is simpler and is more efficient if only the set of formal concepts is considered.

Now we are ready to describe how the new canonicity test can be implemented into the algorithm of the (recursive) *CbO* presented in pseudocode in Algorithm 1. As the above explanation and Proposition 2 show, during the computation we have to propagate the information about sets $B \oslash j$ which take part in the additional test down the call tree, from the root node to the leaves. As a consequence, we have to change the search strategy of

the algorithm since the depth-first search in the space of all formal concepts as it is used in the recursive *CbO* is no longer possible. We modify the procedure GenerateFrom from Algorithm 1 in the following way. We add the additional test before the new formal concept computation (line 7) and instead of the recursive calls after the computation (line 9), the algorithm stores the information about computed formal concepts in a *queue*, in case of successful pass of the original canonicity test (line 8). In case of the test failure, the algorithm updates the required information about the set $B \otimes j$ for propagation down the call tree in a *"propagation" variable* (for each attribute j) passed along in the recursive calls. Then, after all attributes j are processed, the algorithm performs a recursive invocation for each formal concept in the queue with the propagation variable as an additional argument. This effectively changes the order in which new formal concepts are computed because we use here a combined depth-first and breadth-first search in the call tree. But it does not change the order of listing of formal concepts because the listing appears after each recursive call, as in the recursive *CbO*. The need for this change is also further demonstrated in an example in [111].

The above described modifications result in a new algorithm called *FCbO* ("F" stands for "*Fast*") which can be seen as an improved version of (the recursive) *CbO* in that we introduced the new canonicity test which saves redundant computations of formal concepts and thus speed-ups the whole computation. *FCbO* is represented by a recursive procedure FastGenerateFrom, see Algorithm 3, another modification of the procedure GenerateFrom of the recursive *CbO*. For a detailed description of the algorithm, see [71, 111]. Particularly for the use of the *"propagation" variable* consisting of sets M_j which are passed along in the recursive calls of the procedure as sets N_y in the third argument. In order to compute all formal concepts of $\langle X, Y, I \rangle$, the procedure is to be invoked with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$, $y = 0$ and $\{N_y = \emptyset \,|\, y \in Y\}$ as its arguments.

Algorithm 3 lists all formal concepts in $\langle X, Y, I \rangle$, in the same order as Algorithm 1 of the recursive *CbO*, each of them exactly once. The proof of its correctness is elaborated in [111], together with an extensive example demonstrating how the algorithm works from the start to the end.

From the point of view of the worst-case time complexity, *FCbO* has the same asymptotic time delay and overall time complexity as *CbO* (and *PCbO*), i.e. $O(|Y|^3 \cdot |X|)$ and $O(|\mathcal{B}(X, Y, I)| \cdot |Y|^2 \cdot |X|)$, respectively. Namely, in the worst case (in case of $\langle X, Y, I \rangle$ for I being the inequality relation on $X = Y$), *FCbO* can degenerate into (the recursive) *CbO*. But in general, it cannot do worse. Moreover, there are strong indications that on average *FCbO* delivers the results faster than *CbO* (average-case time complexity analysis of *FCbO* and mitigation of the worst-case complexity remain to be challenging and important open problems).

Let us note that *FCbO* can be turned into a "*Fast NextClosure*" algorithm,

Algorithm 3: Procedure FastGenerateFrom($\langle A, , B\rangle, y, \{N_y \,|\, y \in Y\}$)

Input: formal concept $\langle A, B\rangle$, number $y \in Y \cup \{n+1\}$ such that $y \notin B$, sets $N_y \subseteq Y \,|\, y \in Y$
Uses : set Y of attributes, number n of attributes, procedure ComputeClosure

```
1  list ⟨A, B⟩ (e.g., print it on screen);
2  if B = Y or y > n then
3  |  return
4  end
5  for j from y upto n do
6  |  set M_j to N_j;
7  |  if j ∉ B and N_j ∩ Y_j ⊆ B ∩ Y_j then
8  |  |  set ⟨C, D⟩ to ComputeClosure(B, j);
9  |  |  if B ∩ Y_j = D ∩ Y_j then
10 |  |  |  put ⟨⟨C, D⟩, j⟩ to queue;
11 |  |  else
12 |  |  |  set M_j to D;
13 |  |  end
14 |  end
15 end
16 while get ⟨⟨C, D⟩, j⟩ from queue do
17 |  FastGenerateFrom(⟨C, D⟩, j + 1, {M_y | y ∈ Y});
18 end
19 return
```

in much the same way as the recursive *CbO* can be turned into *NextClosure* described in Section 2.2.1. We refer to [111] for details. And recall also that this way for obtaining formal concepts in the lexical order is yet more faster than the iterative *NextClosure* way from [49, 51], due to the employment of the new canonicity test. See performance comparisons in Section 2.5 for illustration.

More (complete) information on *FCbO* can be found in [111]. Let us conclude this section by just a tiny note on parallelization of *FCbO*. In fact, *FCbO* can be turned into a parallel (and distributed) algorithm in the very same way as the recursive *CbO* was turned into *Parallel CbO (PCbO)*, as described in Section 2.2.2 and [73, 74]. The parallel version of *FCbO* obtained this way shall be called *Parallel Fast Close-by-One (PFCbO)* [71].

2.2.4 Computing a single formal concept

To complete the algorithms described in the previous sections we have to add the procedure ComputeClosure common to all of the algorithms. The procedure efficiently computes a new formal concept from an existing one by enlarging its intent by adding an attribute to the intent and shrinking the extent of the concept at the same time. We will now describe the idea behind the algorithm of the procedure.

If $\langle A, B\rangle$ is a formal concept of $\langle X, Y, I\rangle$ then due to the monotony property of $^{\downarrow\uparrow}$, all formal concepts whose intents are strictly greater than B can be written as $\langle (B\cup C)^{\downarrow}, (B\cup C)^{\downarrow\uparrow}\rangle$, where $C \subseteq Y$ is a nonempty set of attributes $j \in Y$ such that there is at least one attribute $j \notin B$. In particular, if we consider $C = \{j\} \subseteq Y$ such that $j \notin B$, then

$$\langle (B \cup \{j\})^{\downarrow}, (B \cup \{j\})^{\downarrow\uparrow}\rangle \tag{2.5}$$

is a formal concept such that $B \subset (B \cup \{j\})^{\downarrow\uparrow}$ and $(B \cup \{j\})^{\downarrow} \subset A$ (by properties of $^{\uparrow}$ and $^{\downarrow}$). This is important from the computational point of view because if we want to compute the extent $(B \cup \{j\})^{\downarrow}$, it is sufficient to go exactly through all objects in A which have in I also attribute j:

$$(B \cup \{j\})^{\downarrow} = \{x \in A \,|\, \langle x, j\rangle \in I\} = A \cap \{j\}^{\downarrow}. \tag{2.6}$$

Similarly, and by the properties of $^{\uparrow}$ and $^{\downarrow}$, the intent $(B \cup \{j\})^{\downarrow\uparrow}$ is formed by the common attributes of objects x from (2.6):

$$(B \cup \{j\})^{\downarrow\uparrow} = (A \cap \{j\}^{\downarrow})^{\uparrow} = \left(\bigcup_{x\in A\cap\{j\}^{\downarrow}} \{x\}\right)^{\uparrow} = \bigcap_{x\in A\cap\{j\}^{\downarrow}} \{x\}^{\uparrow}. \tag{2.7}$$

We have just outlined the idea behind the algorithm which efficiently computes formal concept (2.5) given formal concept $\langle A, B\rangle$ and attribute $j \in Y$ which does not belong to B. The corresponding procedure ComputeClosure is depicted in Algorithm 4. The proof of correctness of the algorithm is presented in [111].

We can plainly see that the worst-case asymptotic time complexity of Algorithm 4 is $O(|X| \cdot |Y|)$. This is clear because we go through each table entry of $\langle X, Y, I\rangle$ at most once. However, in practical situations, the number of table entries we have to go through in order to compute the new formal concept is much smaller than $|X| \times |Y|$.

Note that, in comparison, the conventional, straightforward, methods for computing formal concept (2.5) are based on definitions (1.1) and (1.2) of the concept-forming operators. These methods are implemented by direct two-way algorithm which first computes the extent $(B \cup \{j\})^{\downarrow}$ which is further used to compute the intent $(B\cup\{j\})^{\downarrow\uparrow}$. That means that the input data table is scanned twice. Contrary to that, our procedure ComputeClosure

Algorithm 4: Procedure ComputeClosure($\langle A, B\rangle, j$)

Input : formal concept $\langle A, B\rangle$, attribute $j \in Y$ such that $j \notin B$
Output: formal concept $\langle C, D\rangle$
Uses : set Y of attributes, concept-forming operators $^{\uparrow}$ and $^{\downarrow}$

```
1 set C to ∅;
2 set D to Y;
3 foreach x in A ∩ {j}↓ do
4 |   set C to C ∪ {x};
5 |   set D to D ∩ {x}↑;
6 end
7 return ⟨C, D⟩
```

goes through the table only once, relying on efficient implementation of sets and a single operation on sets: intersection. Since computing set intersections is generally more efficient than implementing the concept-forming operators, Algorithm 4 significantly outperforms the direct two-way algorithm. The efficient implementations of sets of objects and attributes and the intersection operation on the sets are presented in Section 2.3.1.
Moreover, we can easily abort the computation of concept $\langle C, D\rangle$ from Algorithm 4 if it turns out during the computation that the number of objects remaining in A together with those satisfying $\langle x, j\rangle \in I$ is not sufficient to form an extent of a given minimal size, in scenarios where a minimal concept extent size is specified. This enables us, for instance, to compute formal concepts whose intents are the so-called frequent closed itemsets used in association rules mining (mentioned in the introduction Chapter 1). The frequency constraint here means exactly the minimal number of objects in the corresponding formal concept extent.

2.3 Efficiency issues

This section is devoted to issues which considerably affect the real performance of the algorithms. First the implementation issues of the algorithms, namely data representation and used data structures, and second the proper preprocessing of input data before the computation of formal concepts. The actual performance of the algorithms is evaluated in the next section.

2.3.1 Efficient data representation

As was already mentioned in the introduction Section 2.1 the actual performance of any algorithm heavily depends on its implementation. The implementation is then much determined by used data structures. In our implementations of all algorithms presented in this thesis we use 0/1 *arrays*

as basic data structures. Such data representation turned out to be very efficient.

The input data table corresponding to $\langle X, Y, I \rangle$ is represented by a (linearly ordered) set of table rows. The set in this context is represented by a linear array of its elements. By a table row, corresponding to object $x \in X$, we mean the set of attributes $\{x\}^{\uparrow} = \{y \in Y \mid \langle x, y \rangle \in I\}$. Each table row is represented by the characteristic vector of the corresponding set of attributes. And the characteristic vector of a set in this context is represented by a 0/1 linear array, that is, a subset $B \subseteq Y = \{0, 1, \ldots, n\}$ is represented by an $(n+1)$-element linear array b of 1s and 0s such that $b[k] = 1$ iff $k \in B$ and $b[k] = 0$ iff $k \notin B$. In fact, this representation of input data table, further denoted by *table*, gives us a usual representation of a table in a computer by a two-dimensional array which corresponds with the usual table representation of a binary relation (I in our case) in the obvious way. That is, the array *table* is filled with 1s and 0s so that $table[i, j] = 1$ iff $\langle i, j \rangle \in I$ and $table[i, j] = 0$ iff $\langle i, j \rangle \notin I$.
Intents of computed formal concepts, as sets of attributes, are represented just like the table rows, i.e. by characteristic vectors of the sets. Extents, as sets of objects, however, are represented another way. Namely by (linearly) ordered lists of objects in the set. Actually, the first element of the list is the number of objects in the set (extent size) and the remaining elements are the *addresses* (pointers in low-level programming languages) of where input data table rows corresponding to the objects are stored in computer memory, rather than the objects themselves. This representation of extents turned out to be more beneficial and more efficient than the characteristic vector representation, above all in the single formal concept computation procedure ComputeClosure from Section 2.2.4, see below. Furthermore, to further increase the performance, the addresses of objects are stored in the list in the (ascending) order in which input data table rows corresponding to the objects are ordered in the table representation. This enables us to easily obtain the object index.
The data structures have been chosen with the aim to achieve the best possible performance of computing formal concepts by procedure ComputeClosure, described in Section 2.2.4. In particular, the 0/1 arrays representing characteristic vectors of sets of attributes are stored in computer memory as the so-called *bitarrays* which are linear arrays of 32-bit or 64-bit integers (depending on the computer platform), where each bit represents presence/absence of an attribute in a set. The bitarray storage of sets of attributes (input data table rows and concept intents) allows us to quickly compute intersections of the sets, in particular input data table rows $\{x\}^{\uparrow}$ processed in the procedure, by using the bitwise "AND" operation which is commonly implemented as the low-level operation directly in microprocessors and other computing hardware. In addition to that, the used represen-

tation of concept extents as lists of objects allows us to easily go through the objects having instant access to the table row corresponding to an object by its address in the computer memory. That is, for a concept $\langle A, B\rangle$ and an attribute $j \notin B$, $C = A \cap \{j\}^{\downarrow}$ is computed so that we go through all objects x in the list of extent of A and test whether $\langle x, y\rangle \in I$ in *table*. At the same time, we compute D by computing intersections of input data table rows $\{x\}^{\uparrow}$ for all such x.
The above described data structures are in more details discussed in [111, 135] and a detailed comparison of various (other) data structures used for computing formal concepts can be found in [76].

2.3.2 Input data preprocessing

The second efficiency issue that affects the real performance of the algorithms and that we will examine below is the input data preprocessing prior to the actual computation of formal concepts. Namely the ordering of objects and attributes of the input data.

First note that from the point of view of formal concepts and concept lattices themselves, the order of objects and attributes in which they appear in formal context is not essential [51]. Namely, one can reorder objects and attributes in an arbitrary way and both the set of all formal concept and the concept lattice remain the same. What only changes is the order of objects/attributes in extent/intent of formal concepts. From the computational point of view, however, it may happen that certain orderings yield better results, in terms of performance, in conjunction with particular algorithms for FCA, most importantly algorithms for computing formal concepts and concept lattice, than other orderings. From this point of view, in general, it is an important feature of the algorithms of whether their performance depends on the order of objects and attributes in the input formal context. We shall call an algorithm (*permutation*) *resistant* whenever all isomorphic copies (in the usual sense) of the input formal context require the same number of elementary computation steps in order to compute all formal concepts (or the concept lattice). I.e. put in other words, the number of the elementary computation steps is the same no matter how we rearrange rows and columns in the input data table. An elementary computation step here is represented by computation of a single formal concept. One can easily see that, e.g., Lindig's *UpperNeighbor* algorithm [87] is resistant. On the other hand, our algorithms, and algorithms from the CbO-family in general, are not resistant (note that Ganter's *NextClosure* is equivalent to *CbO* in this respect, see also Section 2.2.1). The order of attributes in input formal context has an impact on the performance of the algorithms since the canonicity test is driven by the order of attributes. As a consequence, a different order of attributes can yield different call trees (recall in Sec-

tion 2.2.1) that may have different numbers of nodes. Therefore, it makes sense to consider different orders of attributes because the proper order can further reduce the number of formal concepts that are computed multiple times, thus improving the performance of the algorithms.

Without any further talking, the proper order of attributes for the CbO-family algorithms (including our algorithms) is such that the attributes in input formal context (columns of the data table) are sorted in the ascending order according to their *support*, that is the number of objects having a particular attribute. Formally, attributes $y \in Y$ of a formal context $\langle X, Y, I \rangle$ are sorted in the ascending order according to $|\{y\}^{\downarrow}|$. We call a formal context with attributes sorted in this way an *ordered formal context* [71]. The assertions in [71] then show that for an ordered formal context the canonicity test of both *CbO* and *FCbO* always succeeds for all attribute concepts (concepts generated by a single attribute, in the first level of recursion of the algorithms) provided that all attributes are distinct (i.e., all columns of the input data table are pairwise distinct). The expected impact on the call trees of the algorithms, their numbers of nodes and the number of formal concepts that are computed multiple times, is demonstrated in [71].

The obvious consequence is that in order to achieve better performance of the algorithms, it is desirable prior the computation to reorder the attributes in input data so that the data represent the ordered formal context. Indeed, our empirical experiments presented in [71] have shown that while processing ordered formal contexts, canonicity tests fail less frequently than in case of formal contexts containing inversions (with respect to the order) and with increasing number of inversions the average number of computed formal concepts grows. At the same time, however, due to this one should take into account whether an algorithm operates on the preprocessed data or the original data when evaluating and comparing algorithms for computing formal concepts, see the evaluations in Section 2.5.

Let us note that the above introduced ordering of attributes has already been used in [48]. But the purpose of the ordering there is much different. In [48], the authors need to use this particular ordering of attributes in their parallel version of Ganter's *NextClosure* algorithm to achieve soundness of the algorithm (i.e. each formal concept is listed only once) and do not consider it otherwise. In our case, the ordering is used and further studied for the sake of increased efficiency and for this purpose our finding of the ordering is thus new.

Also, in addition to ordering attributes, objects of input formal context (rows of the data table) can also be ordered. Our experiments with the CbO-family algorithms on contexts of several sizes, densities (percentage of $\times$s or 1s in the table) and origin (real and generated data) have indicated that the performance of the algorithms increases if objects (table rows) are ordered lexicographically according to the characteristic vector of the corresponding set of attributes of the object. The increase is, however, much smaller,

almost negligable, in comparison to the increase of performance in case of the ordering of attributes.

2.4 *Attribute sorting* algorithm

Motivations by the results of attribute ordering presented in the previous section and in [71], and elaborating on the idea, led us to a new algorithm for computing formal concepts. The algorithm is based on attribute sorting and formal context reduction performed after obtaining a new formal concept. I.e., unlike in the previous section where ordering of attributes was just a means of data preprocessing and was used for the input data exactly once before the computation, we utilize the ordering during the computation several times. This results in a conceptually new algorithm which, as we shall see, in terms of the number of formal concepts computed multiple times, outperforms *CbO* and also *Fast CbO* by an order of magnitude. The algorithm will be briefly described in this section, the detailed description can be found in [72].

First we need to introduce basic operations with formal contexts that are used to describe the algorithm. One of the distinguishing features of the algorithm is that during the computation, it transforms the initial formal context into other formal contexts by taking subsets of objects (context reduction operation) and grouping attributes (context clarification operation). In addition to that, the groups of attributes are sorted according to their support and equipped with a Boolean flag indicating whether a group is allowed to be present in intents of formal concepts computed in subsequent stages (as we will see, the flag supports the canonicity test).
In order to keep information about groups of attributes, we use particular formal contexts, called R-contexts, to represent input data. An *R-context* (*derived from formal context* $\langle X, Y, I\rangle$) is a triplet $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ where $X^\sharp \subseteq X$, $Y^\sharp \subseteq 2^Y$ such that any $B_1, B_2 \in Y^\sharp$ are nonempty either equal or pairwise disjoint ($B_1 \cap B_2 = \emptyset$) subsets of attributes from Y where for any $x \in X^\sharp$ and $B \in Y^\sharp$ $\langle x, y_1\rangle \in I$ iff $\langle x, y_2\rangle \in I$ holds true for all $y_1, y_2 \in B$, and $I^\sharp = \{\langle x, B\rangle \in X^\sharp \times Y^\sharp \,|\, \langle x, y\rangle \in I \text{ for all } y \in B\}$. If $X^\sharp = X$, $Y^\sharp = \{\{y\} \,|\, y \in Y\}$, and $I^\sharp = \{\langle x, \{y\}\rangle \in X^\sharp \times Y^\sharp \,|\, \langle x, y\rangle \in I\}$, $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ is called an initial R-context (derived from $\langle X, Y, I\rangle$). Note that each R-context $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ is a well-defined formal context in which attributes have natural interpretation as sets of attributes from the original formal context $\langle X, Y, I\rangle$ which are indistinguishable in $\langle X, Y, I\rangle$ (equal columns in the corresponding data table) provided we restrict ourselves only to objects from $X^\sharp$. Note also that $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ which results from $\langle X, Y, I\rangle$ is fully given by the sets $X^\sharp$ and $Y^\sharp$ of objects and attributes, respectively. The binary relation $I^\sharp$ can be determined from the original binary relation I, thus not needed to be represented in computer memory. See [72] for

other basic properties of R-contexts. Finally, the concept-forming operators $^{\uparrow_{I^\sharp}}$ and $^{\downarrow_{I^\sharp}}$ induced by R-contexts are straightforward restrictions of the concept-forming operators $^{\uparrow}$ and $^{\downarrow}$ of original formal contexts (to $X^\sharp$ and the union of sets $Y^\sharp$ of attributes). The close relationship among them is presented in [72]. An example of R-context is presented in Example 2. From now on we describe further operations with formal contexts in terms of R-contexts instead of the original input formal contexts.

In general, R-context can contain two or more indistinguishable attributes. The algorithm, however, relies on grouping indistinguishable attributes together so that all attributes are distinct (we will see below why). The grouping is done by a process of *clarification* of R-context. Recall from [51] that a formal context $\langle X, Y, I\rangle$ is called *clarified* if for any $y_1, y_2 \in Y$ it follows that $\{y_1\}^{\downarrow} = \{y_2\}^{\downarrow}$ implies $y_1 = y_2$ and dually for objects. Put in words, a clarified formal context in sense of [51] is a formal context where all columns in the corresponding object-attribute data table are distinct and dually for rows. The process of clarification then consists of removing duplicate columns and rows from the table. It is a well known fact that the concept lattice of a clarified formal context is isomorphic to the concept lattice of the original formal context.

Clarification of R-contexts performed by the algorithm applies to attributes of R-contexts only. The basic idea is the same as in [51], we produce a new R-context by putting together identical columns of the corresponding data table. Thus, for any R-context $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$, a *clarified R-context* (which results from $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$) has the same objects $X^\sharp$ and attributes are unions of attributes in $Y^\sharp$ (which are sets of original attributes from Y) for which columns of the data table corresponding to $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ given by the attributes are equal (i.e. $\{y_1\}^{\downarrow_{I^\sharp}} = \{y_2\}^{\downarrow_{I^\sharp}}$ for all pairs of attributes y_1, y_2 in the union). Hence such indistinguishable attributes are grouped together. The relation between the objects and the new attributes is obvious. The exact formal definition of clarified R-context can be found in [72], together with two basic properties (namely that each clarified R-context is a well-defined R-context and that the clarification of a clarified R-context does not change the R-context). An example of clarified R-context is presented in Example 2.

Remark 1 *Notice that we do not consider clarification of objects (i.e., a clarified R-context may contain several objects having the same attributes), since it would not reduce the number of formal concepts computed multiple times and is thus not used in the presented algorithm.*

A crucial operation of the algorithm is *attribute sorting*. In particular, for each R-context $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$, we consider a partial order $\leq^\sharp$ on $Y^\sharp$ such that for any $y_1, y_2 \in Y^\sharp$, $y_1 \leq^\sharp y_2$ implies $|\{y_1\}^{\downarrow_{I^\sharp}}| \leq |\{y_2\}^{\downarrow_{I^\sharp}}|$. I.e., the same ordering of attributes according to their support introduced in the previous

section and further investigated in [71]. Note that, in general, $\leq^\sharp$ is not a linear order but it can be extended to a linear order by a well-known procedure of topological sorting. For the purposes of the algorithm, we identify some (but fixed) of the linear orders with a mapping which assigns to each attribute from $Y^\sharp$ its numerical index which represents a position in an ordered list of attributes which are sorted according to the linear order: $f\colon Y^\sharp \to \{0, \ldots, |Y^\sharp| - 1\}$ such that, for any $y_1, y_2 \in Y^\sharp$,

$$\text{if } f(y_1) \leq f(y_2) \text{ then } |\{y_1\}^{\downarrow_{I^\sharp}}| \leq |\{y_2\}^{\downarrow_{I^\sharp}}|. \tag{2.8}$$

The clarified R-context in Example 2 is presented with attributes sorted according to such a mapping f, in a (usual) way that if $f(y_1) < f(y_2)$ then y_1 is depicted before y_2. (Note that in particular case of the example, there are two ways to define f, since attributes $\{1, 4\}$ and $\{3\}$ have the same support. In such situations, we always consider an arbitrary, but fixed, f for the same R-context.)

Now, as already mentioned in the beginning of this section, an important distinguishing feature of the algorithm is that, unlike in the data preprocessing only approach from the previous section, we do not consider single $\leq^\sharp$ (i.e., a single f) during the computation. The algorithm uses a particular *reduction operation* on R-contexts to reduce the problem of computing formal concepts of an R-context to the problem of computing formal concepts of several smaller R-contexts (the usual *divide et impera* scheme). After each reduction, we determine new f which applies to the reduced R-context. The input for the reduction is an R-context $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$ and the sets C and D of objects and attributes, respectively, of a formal concept $\langle C, D \rangle$ of $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$ whose intent is nonempty, i.e. $D \neq \emptyset$. The output is a sub-context of the R-context with objects taken from C and attributes being attributes in $Y^\sharp$ which are not present in D (cf. the definitions of $X^\mathfrak{R}$, $Y^\mathfrak{R}$, and $I^\mathfrak{R}$ in [72]). One can easily see that this is a well-formed R-context and we denote the clarification of it by $\textsc{Reduce}(\langle X^\sharp, Y^\sharp, I^\sharp \rangle, C, D)$.

Furthermore, we assume that we are given a mapping f which determines the order of attributes in $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$ (see above). Since D is nonempty, we can denote by $\min(D)$ the least attribute from D with respect to the order given by f, i.e., $\min(D) \in D$ such that $f(\min(D)) \leq f(y)$ for all $y \in D$. Each attribute B in the sub-context is then associated with a Boolean flag the value of which is set to *true* if $f(B)$ is less than $f(\min(D))$. Put in words, an attribute $B \in Y^\sharp$ will be given a *true* flag if it is not in D and if it stands before $\min(D)$ in terms of the order of attributes. If B stands behind $\min(D)$, the flag is not updated—attributes of the initial R-context have the flag equal to $false$. The meaning of the flag is that "at least one of the original attributes from B is not permitted to be used (at a certain level of computation)". Compare this with the (new) attributes in the closure D of $B \cup \{j\}$ which are not present in B and are "before j" w.r.t. the order in

which attributes are added to computed formal concepts in the canonicity tests of the recursive *CbO* described in Section 2.2.1 and *Fast CbO* described in Section 2.2.3. Indeed, this is a canonicity test, see the pseudocode of the algorithm below. In this respect it is also important to note that in the clarification of the sub-context, the flag of a new (grouped) attribute, which is a union of attributes in $Y^\sharp$, results by taking the logical "OR" of flags of all attributes in the union. Since all the attributes in the union are indistinguishable within the sub-context. That is why the clarification is necessary and all attributes must be distinct (also cf. the assertions for ordered formal context and canonicity tests of *CbO* and *FCbO* from [71] mentioned in the previous section).

Remark 2 *Note that in the original description of the algorithm in [72] the (numerical) flag is set to the size of B with the meaning "exactly n of the original attributes ..." and in the sub-context clarification sum of flags is taken. This is, however, not necessary for the algorithm since for the canonicity test the only important fact is whether at least one of the attributes in intent D has its flag set.*

The reduction (and clarification) of the R-context from Example 2 by formal concept $\langle C, D\rangle = \langle\{d, e\}, \{\{1, 4\}\}\rangle$ is presented in the same example below.

Example 2 *As an example, consider an input formal context $\langle X, Y, I\rangle$ with objects $X = \{a, \ldots, f\}$, attributes $Y = \{0, \ldots, 7\}$, and I given by the table depicted in the top part of Figure 2.3. An R-context $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ derived from $\langle X, Y, I\rangle$ and the clarified R-context which results from $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ are depicted in the middle part of the figure. Notice that the original attributes 1 and 4 are distinguishable in $\langle X, Y, I\rangle$ by object c. On the other hand, they are indistinguishable on objects $\{b, d, e, f\}$, hence the (group) attribute $\{1, 4\}$ in $Y^\sharp$. During the clarification, only attributes $\{2\}$ and $\{7\}$ have been put together. Note also that attributes in the clarified R-context are sorted according to their support. Finally, a sub-context which can result from reduction (and clarification) of the R-context is depicted in the bottom part of Figure 2.3. Notice that during the reduction, attribute $\{6\}$ was given a true flag–depicted by the column corresponding to the (group) attribute with gray background color–since its position was before that of attribute $\{1, 4\}\rangle$ in the R-context.*

Finally, we can present the algorithm. The main part of it is a recursive procedure COMPUTE the pseudocode of which is depicted in Algorithm 5. The procedure accepts as its argument a clarified R-context and during the computation it calls an auxiliary procedure CLOSURE whose pseudocode is depicted in Algorithm 6.

$\langle X, Y, I\rangle$	0	1	2	3	4	5	6	7
a		×	×		×			
b	×		×	×		×		×
c				×	×			×
d	×	×	×		×	×	×	×
e	×	×			×	×		
f	×		×	×		×		×

$\langle X^\sharp, Y^\sharp, I^\sharp\rangle$	$\{1,4\}$	$\{2\}$	$\{3\}$	$\{6\}$	$\{7\}$
b		×	×		×
d	×	×		×	×
e	×				
f		×	×		×

clarified $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$	$\{6\}$	$\{1,4\}$	$\{3\}$	$\{2,7\}$
b			×	×
d	×	×		×
e		×		
f			×	×

reduction (and clarification) of $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$	$\{3\}$	$\{2,6,7\}$
d		×
e		

Figure 2.3: Formal context (top), derived R-context and its clarification (two middle) and reduced (and clarified) R-context (bottom).

Remark 3 *Note that in the description and the pseudocode of the algorithm in [72], the flag of an attribute B of R-context is (formally) represented by the first item, denoted by n, of a tuple $\langle n, B\rangle$ which actually represents the attribute B. For the sake of reducing the complexity of notation, in the pseudocode of procedure* Compute *below we represent the flag by the notation $B.flag$.*

Briefly, when invoked with (clarified) R-context $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$, procedure Compute first processes the formal concept (e.g., prints it on the screen or stores it) which consists of the set of objects $X^\sharp$ and the set of attributes $Y \setminus \bigcup\{B \subseteq Y \mid B \in Y^\sharp\}$ (cf. the notation $\textsc{Int}(\mathbb{K}^\sharp, Y)$ from [72]). I.e. all objects of the R-context and attributes which are not present in any (group) attribute of the R-context (recall above how a sub-context of R-context is created). Then, the procedure goes over all attributes in $Y^\sharp$ with $false$ flag and for each such attribute invokes procedure Closure. An easy inspection of the pseudocode in Algorithm 6 shows that the result of calling $\textsc{Closure}(\langle X^\sharp, Y^\sharp, I^\sharp\rangle, B)$ is the formal concept of R-context $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ generated by attribute B, i.e., $C = \{B\}^{\downarrow_{I^\sharp}}$ and $D = C^{\uparrow_{I^\sharp}}$. Notice that Algorithm 6 utilizes attribute sorting together with the fact that $\langle X^\sharp, Y^\sharp, I^\sharp\rangle$ is clarified. In that case, all attributes which belong to D must have their

Algorithm 5: Procedure COMPUTE($\langle X^\sharp, Y^\sharp, I^\sharp \rangle$)

```
Input: R-context ⟨X♯, Y♯, I♯⟩
Uses : set Y of attributes, procedures CLOSURE and REDUCE
1  list ⟨X♯, Y \ ⋃{B ⊆ Y | B ∈ Y♯}⟩ (e.g., print it on screen);
2  for B ∈ Y♯ do
3  |  if B.flag = false then
4  |  |  set ⟨C, D⟩ to CLOSURE(⟨X♯, Y♯, I♯⟩, B);
5  |  |  if ⋁{B.flag | B ∈ D} = false then
6  |  |  |  COMPUTE(REDUCE(⟨X♯, Y♯, I♯⟩, C, D));
7  |  |  end
8  |  end
9  end
10 return
```

indices strictly greater than or equal to $f(B)$. This observation has already been made in [71]. The formal concept $\langle C, D \rangle$ then undergoes the canonicity test which succeeds iff the flags of all attributes in D are $false$ (recall the meaning of the flags above). In case of success, COMPUTE just invokes itself with reduced (and clarified) formal context which results from $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$ by C and D.

In order to compute all formal concepts of formal context $\langle X, Y, I \rangle$, procedure COMPUTE has to be invoked with REDUCE($\langle X^\sharp, Y^\sharp, I^\sharp \rangle, X, X^{\uparrow_{I^\sharp}}$) as the argument, where $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$ is the initial R-context derived from $\langle X, Y, I \rangle$. Put in words, COMPUTE has to be invoked with the clarification of the initial R-context derived from $\langle X, Y, I \rangle$ without attributes shared by all objects ($X^{\uparrow_{I^\sharp}}$)—the procedure then stores (as the first step) the first formal concept consisting of all objects and those attributes. The proof of soundness of the algorithm, i.e. that with such input data it lists all formal concepts, each of them exactly once, is provided in [72]. [72] also includes an illustrated full running example demonstrating the behavior of procedure COMPUTE.

The asymptotic worst-case time complexity of Algorithm 5 is the same as in case of *CbO* and *FCbO* algorithms (Sections 2.2.1 and 2.2.3), i.e., $O(|\mathcal{B}(X, Y, I)| \cdot |X| \cdot |Y|^2)$. See [72] for a brief analysis. In case of time delay [63], the algorithm has the same polynomial time delay $O(|Y|^3 \cdot |X|)$ as *CbO*, cf. [84]. The argument remains the same as in case of *CbO*.

More interesting is comparison of the algorithm with *CbO* and *FCbO* in terms of formal concepts which are computed multiple times. Figure 2.4 shows a call tree for both *CbO* and *FCbO* (applied to input formal context from the running example in [72]). Recall that a call tree depicts computations of *FCbO*/*CbO* where black/gray square leaf nodes labeled by formal concepts represent branches of computation where the concepts are computed but fail the canonicity test. The black nodes and bold edges cor-

Algorithm 6: Procedure CLOSURE($\langle X^\sharp, Y^\sharp, I^\sharp \rangle, B$)

Input : clarified R-context $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$, attribute $B \in Y^\sharp$
Output: formal concept $\langle C, D \rangle$
Uses : mapping f determining the order of attributes in $\langle X^\sharp, Y^\sharp, I^\sharp \rangle$

```
1  C = {x ∈ X♯ | ⟨x, B⟩ ∈ I♯};
2  D = {y ∈ Y♯ | f(B) ≤ f(y)};
3  for x ∈ C do
4  |  for y ∈ D do
5  |  |  if ⟨x, y⟩ ∉ I♯ then
6  |  |  |  remove y from D;
7  |  |  end
8  |  end
9  end
10 return ⟨C, D⟩
```

respond to both *CbO* and *FCbO* while the gray nodes and dotted edges correspond only to *CbO*. We can see from the tree that *FCbO* computes 7 formal concepts which fail the (additional) canonicity test and are thus computed multiple times and for *CbO* the number of computed concepts which fail the canonicity test is even 19 (the number of formal concepts of the corresponding formal context is 11). Algorithm 5 computes for the corresponding input formal context just a single formal concept twice, namely R_3 (its R-context pre-image, to be precise), i.e. commits just a single canonicity test fail! This is a really significant improvement. Section 2.5 below then shows a brief experimental evaluation of average behavior of Algorithm 5 compared to *CbO* and *FCbO* using various data sets which shows an interesting tendency that the numbers of formal concepts computed multiple times by the algorithm are much smaller.

2.5 Experimental evaluation

In this section we briefly illustrate the performance of the above described algorithms and compare them with other algorithms for computing formal concepts [84, 126]. More rigorous performance evaluations can be found in the papers on the algorithms from which the following results are borrowed.

We show the results from three experiments. In the first two of them we were interested in the performance of algorithms measured by running time and compared the performance of the recursive *CbO*, *PCbO* and *FCbO* algorithms with algorithms in literature commonly used as referential, Ganter's *NextClosure* [49, 51] and Lindig's *NextNeighbor* [87], and also to Berry's al-

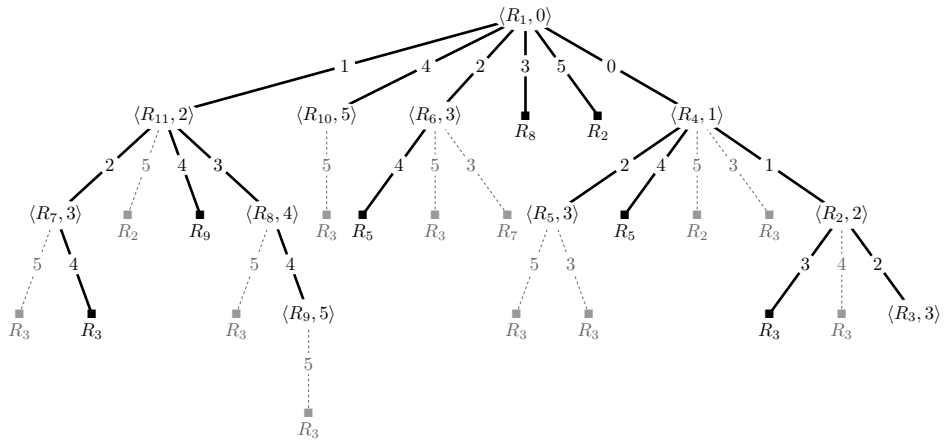

Figure 2.4: An example of call tree for *CbO* and *FCbO*.

gorithm [31]. In the third experiment we measured and compared the total numbers of formal concepts computed by (recursive) *CbO*, *FCbO* and the *attribute sorting* algorithm. The experiments were done on several public real-world benchmark datasets from the UCI Machine Learning Repository [6], the UCI Knowledge Discovery in Databases Archive [61] and also our own dataset describing software packages in the Debian GNU/Linux operating system distribution.

In the first experiment, borrowed from [111], we in addition compared *FCbO* and (recursive) *CbO* also in terms of the total number of computed formal concepts, in order to evaluate the influence of the new canonicity test introduced in *FCbO* (see Section 2.2.3). The results are depicted in Table 2.1, along with the information on size and *density* (percentage of ×s or 1s in the dataset table) of used datasets and the number of formal concepts in the datasets. Note that in this experiment we applied the preprocessing step of ordering attributes of the dataset table according to their support prior to computation, as described in Section 2.3.2, in order to further lower the total number of computed formal concepts (#closures in Table 2.1). The numbers for the case without the ordering of attributes are included in results of the third experiment below.

First note that both (recursive) *CbO* and *FCbO* significantly outperform the *NextClosure* algorithm. The huge performance gain is due to (1) the different order of computed formal concepts and (2) more efficient computation of formal concepts, described in Section 2.3.1. Next, we can see that *FCbO* outperforms (recursive) *CbO*, both in terms of the number of computed formal concepts and the running time. Here, the performance gain is due to the new canonicity test which avoids a large number of formal concepts to be computed multiple times, see the numbers of concepts computed by *FCbO* and *CbO* in the table. *FCbO* is typically faster than *CbO*, since the total

Table 2.1: Performance (in seconds) and total numbers of formal concepts computed by *CbO* and *FCbO* for selected datasets.

dataset	mushroom	anonymous web	adult	internet ads
size	8124 × 119	32710 × 295	48842 × 104	3279 × 1557
density	19.33 %	1.02 %	8.65 %	0.88 %
#concepts	238710	129009	180115	9192
NextClosure time	53.891	243.325	134.954	114.493
CbO time	0.508	0.238	0.302	0.332
FCbO time	0.340	0.240	0.318	0.160
#closures computed by *CbO*	1321524	785394	585253	1783871
#closures computed by *FCbO*	299201	398147	305644	309357

Table 2.2: Performance of *PCbO* and other algorithms for selected datasets (real running time, in seconds; time in parentheses represents total running time used by all processes together).

dataset	mushroom	tic-tac-toe	Debian tags	anonymous web
size	8124 × 119	958 × 29	14315 × 475	32710 × 295
density	19 %	34 %	< 1 %	1 %
PCbO ($P = 1$)	4.89	0.06	7.79	40.32
PCbO ($P = 2$)	2.78 (5.16)	0.04 (0.07)	5.52 (9.34)	22.16 (43.33)
PCbO ($P = 4$)	1.90 (5.39)	0.03 (0.07)	3.65 (10.88)	13.38 (47.81)
PCbO ($P = 8$)	1.18 (5.58)	0.02 (0.07)	2.51 (11.08)	8.09 (46.68)
NextClosure	834.40	2.15	1720.82	10039.73
NextNeighbor	5271.98	14.53	2639.67	13422.64
Berry's	934.50	5.78	1531.94	3615.07

number of computed formal concepts directly influences the performance of the algorithms. But notice that in the worst case *FCbO* collapses into *CbO* (cf. Section 2.2.3). For further experiments on the impact of the new canonicity test in *FCbO* (e.g. frequency/rate of successful tests) and its performance comparison to *CbO* and other well-known algorithms on both real and artificial (randomly generated) datasets, see [71, 111].

In the second experiment, borrowed from [74], we focused mainly on scalability of the *PCbO* algorithm, i.e. the growth of the algorithm's performance (in terms of decrease of running time) with respect to the growing number of processes used for computing formal concepts in parallel. The results are depicted in Table 2.2 and Figure 2.5, again along with the information on size and density of used datasets.

We can see that *PCbO* outperforms the other compared algorithms by several magnitudes. This is not surprising. More interesting are the *PCbO* alone times regarding the scalability. The first four rows in Table 2.2 contain running times of *PCbO* that has been run in 1 (which equals the sequential version, recursive *CbO*), 2, 4, and 8 processes. We can see the expected speedup (decreasing running time) of the algorithm when increasing the number P of processes. Furthermore, for $P > 1$, the rows contain also total running time, written in parentheses, used by all processes together to compute all formal concepts. This time allows us to make a rough estimate of

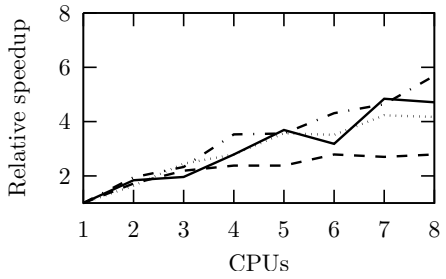

Figure 2.5: Relative speedup of *PCbO* to the number of processes for selected datasets (solid line—mushroom, dashed line—tic-tac-toe, dotted line—Debian tags, dot-and-dashed line—anonymous web).

the overhead that is needed to manage the (multiple) processes: the overhead can be computed as the real running time minus the total running time divided by P. As expected, larger values of P lead to larger overhead.
For the sake of illustration, we include also a graphical depiction of the speedup, in Figure 2.5 (also borrowed from [74]). By a *relative speedup* which is shown on y-axis of the graph in the figure we mean the theoretical speedup given by the number of processes (e.g., if we have 4 processes, the execution can be 4 times faster). Therefore, the relative speedup is a ratio of running time using a single process and running time using multiple processes. Note that the theoretical maximum of speedup is equal to the number of used processes but real speedup is always smaller due to the overhead needed to manage the processes (cf. also Table 2.2). Nevertheless, from the point of view of the speedup, we can see from the graph that the real speedup of the *PCbO* algorithm is near its theoretical limits. See [73, 74] for further experiments on the scalability, speedup and overhead of *PCbO* (and *PFCbO*, in [71]), as well as the utilization of processes, i.e. the workload distribution schemes and numbers of formal concepts computed by processes, on both real and artificial (randomly generated) datasets.
The third experiment again focuses on the total number of computed formal concepts since, as seen above in the first experiment, it is a feature significantly affecting performance of all algorithms in the CbO-family. In this experiment, however, Table 2.3 shows the numbers, besides for the (recursive) *CbO* and *FCbO* algorithms, also for the *attribute sorting* algorithm (Algorithm 5 from Section 2.4). Note that for results of *CbO* and *FCbO* the table contains two rows: the rows labeled "(ordered)", as opposed to the rows without this label, present the numbers for the case when the additional preprocessing step of ordering attributes of input data table according to their support is applied prior to computation (cf. Section 2.3.2).
First of all, it follows from the table that the *attribute sorting* algorithm needs to compute considerably less formal concepts than the other algorithms, namely *CbO* and also *FCbO*. Apparently, the new method of com-

Table 2.3: Total numbers of formal concepts computed by *CbO*, *FCbO* and *attribute sorting* algorithm for selected datasets.

dataset	Debian tags	anonymous web	mushroom	tic-tac-toe
size	14315 × 475	32710 × 295	8124 × 119	958 × 29
density	< 1 %	1 %	19 %	34 %
#concepts	38977	129009	238710	59505
attribute sorting	44221	135925	246181	65567
FCbO (ordered)	298641	398147	299201	89930
FCbO	679911	1475341	426563	128434
CbO (ordered)	960106	785394	1321524	185738
CbO	12045680	27949552	4006498	221608

puting formal concepts can reduce the total number of computed formal concepts by several orders of magnitude. The factor of improvement depends on many aspects, notably the size and density of input data. In case of large and sparse datasets like *anonymous web* and *Debian tags*, the algorithm needs to compute only a small fraction of concepts multiple times—in a strong contrast to *CbO*, in particular. As for the preprocessing of input data by ordering attributes according to their support, Table 2.3 confirms (for *CbO* and *FCbO*) that this action alone also lowers the numbers of formal concepts computed multiple times, as discussed in Section 2.3.2. The differences in total numbers of computed formal concepts are apparent again for large and sparse datasets and, obviously (due to the new canonicity test), are much larger for *CbO* than for *FCbO*. Note that these tendencies, both for the *attribute sorting* algorithm and the input data preprocessing step only, are quite general. For the algorithm, they are further illustrated in [72] on artificial (randomly generated) data of fixed size with various densities and data with growing number of attributes (interestingly, the number of objects has no noticeable impact).
Further experiments on various aspects of the particular algorithms based on (the recursive) *CbO* can be found in [71].

2.6 Summary and topics for future research

We have summarized new algorithms for computing formal concepts from object-attribute relational data. From the many algorithms developed in the past and known from the literature our algorithms differ by their performance efficiency. The algorithms, with the exception of the last one, are based on a recursive version of Kuznetsov's *Close-by-One (CbO)* [80, 79, 81] algorithm and as such they use a so-called canonicity test to ensure that, while computed multiple times, formal concepts are listed exactly once, in a unique order. The order is more efficient than the lexical order used by Ganter's *NextClosure* algorithm [49, 51]. We call algorithms using the *CbO*'s canonicity test, including our new algorithms, the CbO-family algorithms.
The recursive version of *CbO* [135], upon which the algorithms summarized

in the previous sections are based, has been presented in Section 2.2.1. *CbO* (without the tree of computed formal concepts from the original description) was formulated as a recursive procedure which is much closer to the actual implementation than the original description of the algorithm in [81] using backtracking.
Then, in Section 2.2.2 we have presented a parallel version of the recursive *CbO*, *Parallel CbO (PCbO)*, which can be run in multiple independent processes on multiple processor cores or processors. The computation of formal concepts is parallelized in such a way that disjoint sets of them can be computed simultaneously (independently) with virtually no overhead (requiring no synchronization) and without increasing the overall asymptotic time complexity of the algorithm. This indeed has a positive impact on performance and scalability of the algorithm. With growing number of processes, the speedup of the computation is near its theoretical limit.
Finally, in Section 2.2.3 we have described an algorithm, called *Fast CbO (FCbO)*, which improves the recursive *CbO* by introducing a new canonicity test. The first part of the test is performed before a new formal concept is computed, eliminating thus the computation of formal concepts for which the original canonicity test in *CbO*, the second part of the test, would fail. The new test, while maintaining virtually neglecting overhead, again does not increase the overall asymptotic time complexity of the algorithm. However, compared to *CbO*, *FCbO* significantly reduces the number of computed formal concepts due to the new test and hence delivers results faster than *CbO*. Furthermore, the same way the recursive *CbO* was turned into *PCbO*, *FCbO* can be turned into a parallel algorithm, resulting in the *PFCbO* algorithm.
To complete the descriptions of the previous algorithms, we added a fast procedure for computing a single (the new) formal concept from another one, the critical procedure for the algorithms, in Section 2.2.4. We took advantage of efficient bitwise level data representation of input data and intents of formal concepts to have the procedure as efficient as possible. The implementation issues, bringing a cutting-edge performance of the algorithms, have been clarified and addressed in Section 2.3.1.
We have also seen, in Section 2.3.2, that the overall performance efficiency of the algorithms can be further increased by preprocessing input data prior to actual computation by sorting and processing attributes in a proper order. Namely, if processing input data with attributes sorted in the ascending order according to their supports, i.e., the number of objects having the attribute, canonicity tests of the algorithms tend to fail less frequently than in case of data containing violations to this order. This indeed results in decreasing the number of formal concepts computed multiple times.
Motivated by this observation, extended to successive attribute ordering (i.e., not just once before the computation) and reduction of the processed part of input data after each formal concept computation and listing, and

combined with basic ideas of (the recursive version of) the *CbO* algorithm, we came up with a conceptually novel approach to computing formal concepts. The algorithm exploiting it, *attribute sorting* algorithm, has been presented in Section 2.4. In terms of the number of formal concepts computed multiple times, the algorithm further significantly outperforms the algorithms from the *CbO* family, including *CbO* and even *Fast CbO*. The number reduces down to a small fraction of the total number of computed formal concepts, keeping the overall theoretical time complexity of the algorithm the same as of the other CbO-family algorithms.
Some results from experimental evaluations of the performance and other aspects of the summarized algorithms have been presented in Section 2.5, with references to papers for more evaluations. Results from the evaluations (not only those included but all we performed) show that our algorithms outperform almost all other algorithms for computing formal concepts from the literature, often by magnitudes, even if well implemented. While those algorithms can process in reasonable time (up to an hour on contemporary commodity computers) data of size up to thousands to tens thousand of objects and a hundred of attributes, our algorithms, properly implemented, allow to process in reasonable time data of size one factor larger, i.e. going to tens to hundreds thousand of objects and hundreds to thousands of attributes. In fact, a single performance competitor algorithm up to date, to our knowledge, is the *InClose2* algorithm [4] recently developed by Andrews. Let us also note that the *FCbO* algorithm, described in Section 2.2.3, became the winner in a formal concepts computing performance competition held within the conference *ICCS 2009*, one of the main conferences devoted to FCA, in Moscow in 2009.
Our performance tuned implementations of all the algorithms, recursive *CbO*, *PCbO*, *FCbO*, *PFCbO*, and the *attribute sorting* algorithm, including all above described efficiency-related improvements, can be downloaded from

`fcalgs.inf.upol.cz`.

There are many topics for future research which are outlined in our papers. Here we list only the most interesting:

- optimizations of the algorithms and, in particular, used data structures for sparse input data (density less than 5 %), which are common for real datasets,

- mitigation of the worst-case time complexity estimation and the average-case time complexity analysis of the algorithms,

- incremental (update) variants of the algorithms, i.e. variants computing (updating) the set of formal concepts of input data that grows

(object by object); actually, we already have the most recent results on this topic, see [106, 107],

- extending the algorithms to compute the cover relation on the set of formal concepts, i.e. the concept lattice, of input data; also on this we already have the most recent results in [106, 107],
- (more) performance comparisons with various recently developed algorithms for computing formal concepts and concept lattice, in particular *InClose2* [4] and *Addintent* [93],
- special variants of the algorithms focused to solve particular problems related to FCA, e.g. factorization of binary (Boolean) matrices in *Boolean matrix factorization (BMF)* [27]; actually, BMF by means of FCA is utilized in Section 3.2 in Chapter 3 and the algorithm for computing factors used there is such a variant (uses the fast procedure for computing a single formal concept and the implementation uses the performance efficient data structures),
- generalizations of the algorithms for data with more general attributes than binary, e.g. graded (fuzzy) attributes.

The algorithms for computing formal concepts described in this chapter have been presented at the main conferences devoted to FCA: *CLA 2008* (PCbO), *ICCS 2009* (FCbO) and *CLA 2010* (FCbO, PFCbO), with publications in the conference proceedings. Extended versions of the respective papers have been published in the *Annals of Mathematics and Artificial Intelligence* (PCbO, attribute sorting algorithm) and *Information Sciences* (FCbO).

Chapter 3

Applying formal concepts

3.1 Inducing decision trees via formal concepts

3.1.1 Introduction

In the second main part of the thesis we present two applications of formal concepts and FCA, the first one in the area of classification of data, in particular decision tree induction.

Decision trees and their induction is one of the most important and thoroughly investigated methods of machine learning [43, 120, 130]. Machine learning is one of the major fields in artificial intelligence which concerns with the development of methods and techniques that allow machines to "learn". Decision trees, being an efficient and most often used classification models of data (with any type attributes), support machine learning in the problem of decision making. A decision tree is typically used for a classification of objects of data into a given set of *classes* based on attributes of the objects. Due to this task, decision trees have also more descriptive names of classification trees or regression trees in the case of discrete or continuous class labels, respectively. For decision tree induction, or construction, many algorithms have been proposed in the literature, see e.g. [122, 130] for an overview. The best-known and most applied algorithms, *ID3* and *C4.5* [120, 121], use local information about objects and their given classes to decide which objects will be covered by a tree node being created in each step during the construction of the tree.
This section is devoted to a novel method of decision tree induction from data with binary attributes (or any type after a transformation to binary, see a note below) utilizing certain formal concepts of input data as nodes of the decision tree constructed from the data. Using formal concepts as nodes of a decision tree is a straightforward idea because both formal concepts and decision tree nodes represent collections (clusters) of objects in input data defined by having the same values for certain attributes. A challenge,

however, consists in how to select the right formal concepts for decision tree nodes. Namely, one cannot directly use all formal concepts in the input data along with their hierarchy, i.e. the concept lattice of the data (obviously) without the least formal concept, as a decision tree induced from the data, just because the concept lattice (without the least element) is not a tree, in general. Nevertheless, one can attempt to consider the concept lattice (without the least formal concept) as a collection of overlapping trees (see [15, 16] for results on input data properties for the concept lattice without the least element to be a tree). The selection of the formal concepts, and thus the problem of construction of a decision tree, then can be reduced to the problem of selection of one of those trees. Our method is, conceptually, based on this idea, but, contrary to the cover relation on formal concepts which usually is the output of algorithms for computing concept lattice, we use a (partial order) relation on the set of computed formal concepts which is in general larger than the cover relation.

To compute the formal concepts and the partial order relation we can use a modified *CbO* algorithm described in Section 2.2.1, 2.2.2 or 2.2.3 with all its advantages described in other sections of Chapter 2 (though, in [14], on which this section is based, we use a modified Lindig's *NextNeighbor* algorithm). Experimental evaluation of our method indicates good classification performance of the method, in that it compares to standard decision tree induction and machine learning methods, outperforming some of them. The method is briefly described in Section 3.1.3. Selected results from the experimental evaluation and comparison with the decision tree induction and machine learning methods like *ID3* or *C4.5* on public real-world benchmark datasets is included in Section 3.1.4. For a more detailed description and more experiments, see [13, 14, 109].

It is important to note that the whole approach of utilizing formal concepts, concept lattices and other instruments of FCA in machine learning and classification, in particular, is not new. Means of FCA have already been in various ways proposed in several machine learning methods in the literature. A well-known approach utilizing particular formal concepts of input data is described in [82] which presents a model of learning from positive and negative examples. Another approach of selecting neighbor formal concepts in concept lattice for classification of unknown objects is presented in [62]. Other approaches are presented, for instance, in [34], describing *GALOIS*, a clustering method based on concept lattices, or in [92], where the authors use FCA in their *IGLUE* method for selection and transformation of attributes which are then used to solve a decision problem by k-nearest neighbor clustering. See [47] for a survey and comparison (in theory and experiments) of several FCA-based classification algorithms which are commonly called *lattice-based* or *concept-based learning* techniques in data mining [45, 113]. According to these attempts the approach of utilizing formal concepts and concept lattices in classification and machine learning in

general seems promising.
Compared to these approaches, the main novelty of our method is in the utilization of the closure properties of formal concepts and the partial order relation on the set of formal concepts directly in the process of construction of the decision tree. Unlike as a preprocessing step (prior to decision tree construction, for instance) or a basis for a new machine learning method. As discussed above, formal concepts and their order have many in common with decision tree nodes and edges.

3.1.2 Preliminaries in decision trees

Before going to the description of our decision tree induction method, let us very briefly recall decision trees basics. More thorough introduction to decision trees and their induction can be found in any literature cited in the end of this section.

In theory, a *decision tree* can be considered as a tree representation of a function over variables which takes a finite number of values. The function is partially described by assignment of function values to vectors of values of the variables. In decision trees, function values are called *class labels*, variables are called attributes and the vectors of values of variables represent records which we identify with objects. Such an assignment is usually represented by a (data) table with rows corresponding to objects (records), columns corresponding to attributes (variables) and for each object containing values of attributes and the class label (function value) assigned to the object (usually given in the last column). For example, the data table (table rows) in Figure 3.1 (top) partially describes a function $f : A \times B \times C \to D$ over three variables A, B and C which take values good and bad (A), yes and no (B) and true and false (C), respectively. The set D of function values consists of values yes and no. The decision trees depicted in Figure 3.1 (bottom) represent two functions, both of which are extensions of f.
In common depictions of decision trees as in Figure 3.1, each non-leaf tree node of a decision tree is labeled by a (circled) attribute, called *splitting attribute* for this node. Such node represents a test, according to which the collection of objects covered by the node (all objects of data for the tree root node) is split into v sub-collections which correspond to v possible outcomes of the test. In the basic setting, the outcomes are represented by values of the splitting attribute, thus a sub-collection contains objects having the particular splitting attribute value. Tree edges connecting the node with nodes corresponding to the sub-collections are labeled by the values. Finally, leaf nodes of the tree, labeled by a (rectangled) class label, represent collections of objects all of which, or the predefined majority of which, have the (same) class label (the latter is a common practice used to avoid the problem of insufficient generalization and "overfitting" of the tree,

A	B	C	$f(A,B,C)$
good	yes	false	yes
good	no	false	no
bad	no	false	no
good	no	true	yes
bad	yes	true	yes

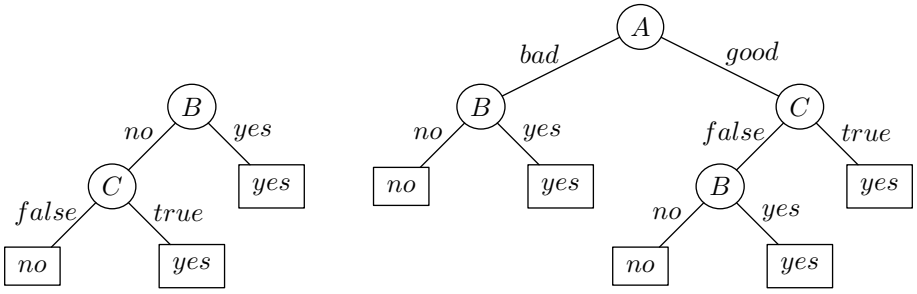

Figure 3.1: Decision trees (bottom) representing functions which are extensions of the function f (top).

see below).

Having a data table partially describing an unknown function, the goal is to construct a decision tree that approximates the function with a desired accuracy. This means that for an object described by its attribute values, the class label assigned by the decision tree to the object is the class label assigned to the object by the function. This is called a *decision tree induction problem*. In the example above, both decision trees assign right, i.e. according to the function they approximate, class labels to all objects in the table. Thus, in practice, decision trees are used to classify objects into classes based on class labels of the objects. A good decision tree is, however, supposed to classify correctly not only the objects described by the input data table, but also previously, during the decision tree induction phase, "unseen" objects—that means to provide a good generalization of classification. Commonly, the input data table is called a *training data set*, and the data table containing the unseen objects a *testing data set*. The training data set is used to induce a decision tree and the testing data set is then used to evaluate the performance and further use of the induced decision tree. To provide a good generalization and avoid the problem of "overfitting" [98] ("overlearning"), where the induced decision tree classifies well (perfectly) the training data set but poorly the testing data set, the aim is to induce a minimal possible tree (in the number of tree nodes) among those which correctly classify the training data set, leaving room for the generalization. The preference of smaller trees also intuitively follows from the so-called Occam's Razor principle according to which the best solution

from equally satisfactory ones is the simplest one.
Many algorithms for induction of decision trees were proposed in the literature, see e.g. [100, 121, 122, 130]. A strategy commonly used consists of constructing a decision tree in a top-down fashion, from the root node to the leaves, by successively splitting existing nodes and creating new ones. I.e., following the description of decision trees above, in basic setting, for every node, a splitting attribute is chosen to split the collection of objects covered by the node into the sub-collections which correspond to values of the splitting attribute. For every such value, a new node is then attached as a child to the node for which the splitting attribute has been chosen. The process continues recursively until all objects corresponding to any (leaf) node, or a predefined majority of them, belong to the same class. A critical point in this strategy is the selection of splitting attributes. There have been proposed many approaches for the selection. These include the well-known and most often implemented approaches based on entropy measures, Gini index and (mis-)classification error, implemented in the best-known decision tree induction algorithms *ID3* and *C4.5* [120, 121], or other measures defined in terms of the class distribution of objects before and after splitting, see [100, 121, 122, 130] for overviews.

3.1.3 Decision tree induction method

In this section we summarize our method of decision tree induction. For details, in particular a full description of the algorithm of the method with a pseudocode of the algorithm, we refer to [14].

But before delving into the own description of the method, let us make a small note on input data (attributes) type and its transformation. In machine learning and classification (and in decision trees at particular), the input data attributes are of various types, very often categorical one. To utilize Formal concept analysis (FCA) with the input data, we need first to transform the (categorical) attributes to binary attributes because, in its basic setting, FCA works with binary attributes. A transformation of input data which consists in substituting non-binary attributes by binary ones which we use is the conceptual scaling [51], mentioned in the introduction to Chapter 1. Obviously we need not transform the class labels assigned to objects in input data because we compute formal concepts over attributes only.
To illustrate the decision tree induction method described below, we will use the input data from Figure 3.2 (top), borrowed from [14]. The data table contains sample animals described by attributes *body temperature*, *gives birth*, *fourlegged*, *hibernates*, and *mammal*, with the last column containing class labels assigned to the animals. After an obvious transformation (nominal scaling) of the attributes, we obtain the data depicted in Figure 3.2

animal	*body temp.*	*gives birth*	*fourlegged*	*hibernates*	*mammal*
cat	warm	yes	yes	no	yes
bat	warm	yes	no	yes	yes
salamander	cold	no	yes	yes	no
eagle	warm	no	no	no	no
guppy	cold	yes	no	no	no

animal	*bt cold*	*bt warm*	*gb no*	*gb yes*	*fl no*	*fl yes*	*hb no*	*hb yes*	*mammal*
cat	0	1	0	1	0	1	1	0	yes
bat	0	1	0	1	1	0	0	1	yes
salamander	1	0	1	0	0	1	0	1	no
eagle	0	1	1	0	1	0	1	0	no
guppy	1	0	0	1	1	0	1	0	no

Figure 3.2: Input data table (top) and corresponding data table for FCA (bottom).

(bottom). In the following, the data will be in an obvious way formally represented by a formal context $\langle X, Y, I \rangle$ (see Section 1.1). Formal concepts utilized in the method are computed from data which we obtain after such transformation and discard the class labels.

Step 1 – computing a partially ordered set of formal concepts
We can now approach the first step of our method of decision tree induction—computing (and storing) formal concepts from input data and determining our partial order relation on the concepts which we will use for the constructed decision tree. Recall that in a decision tree nodes cover some collection of objects which is split creating other nodes which cover smaller collections of objects. Similarly, for formal concepts and the partial order relation on them modeling the subconcept-superconcept hierarchy (i.e. in a concept lattice, cf. (1.3) in Section 1.1), smaller concepts (subconcepts) result by adding attributes to (the intent of) larger concepts (superconcepts) and, due to this refinement, smaller concepts cover smaller collection of objects than larger concepts. Thus, for constructing a decision tree from input data in a common top-down fashion we need an algorithm which iteratively computes smaller formal concepts from a larger formal concept, starting with the largest one (which covers all objects). Moreover, in a decision tree collections of objects covered by nodes are split until the covered objects, or the predefined majority of the objects, have the same assigned class label. Hence, when computing smaller formal concepts we also need not refine formal concepts which cover those objects (or the predefined majority of them) that have the same class label.
Such an algorithm, which we used in [13, 14], is for instance the well-known Lindig's *NextNeighbor* [87] algorithm for computing concept lattice. The algorithm is used in [13, 14] with the following required modifications. First,

as noted above, we do not compute smaller formal concepts from a formal concept which covers (whose extent contains) objects, or the predefined majority of the objects, that have the same class label. Second, contrary to the original *NextNeighbor* algorithm which in its top-down version computes, besides formal concepts, the cover relation on formal concepts, in our modification we compute a partial order relation on the set of computed formal concepts which is in general larger than the cover relation. In cover relation, formal concepts $\langle A, B\rangle$ (a larger one) and $\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\rangle, y \in Y$ (a smaller one) of $\langle X, Y, I\rangle$ need not be related (this happens if there is a concept "in between" covered by $\langle A, B\rangle$ and covering $\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\rangle$). As mentioned above, formal concepts correspond to nodes of the constructed decision tree in our approach. Let $y_v \in Y$ be a binary attribute corresponding to value v_a of a (categorical) attribute a of the original input data, cf. the transformation of input data above. In our modified relation we need to relate with formal concept $\langle A, B\rangle$ all the formal concepts $\langle (B \cup \{y_v\})^{\downarrow}, (B \cup \{y_v\})^{\downarrow\uparrow}\rangle$, for all y_v, in order to keep the possibility of having nodes n_{y_v} corresponding to the concepts, respectively, in the resulting decision tree. If a is the splitting attribute for node n corresponding to $\langle A, B\rangle$ in the decision tree, then $\langle (B \cup \{y_v\})^{\downarrow}, (B \cup \{y_v\})^{\downarrow\uparrow}\rangle$ is the formal concept corresponding to node n_{y_v} which is connected to n in the tree via an edge. The concept results as an outcome of the test "what is the value of a?" (the outcome is represented by value v_a of the splitting attribute a). Interestingly, with such modification the *NextNeighbor* algorithm becomes the recursive *CbO* algorithm described in Section 2.2.1 in which we refrain from adding attributes in a fixed order and use instead of the *CbO* canonicity test (cf. (2.1) in the section) the *NextNeighbor*'s canonicity test (i.e. looking only for the presence of computed formal concepts in a data structure where the concepts are stored). As a "bonus", by using such a modified (recursive) *CbO* we can afford some of the advantages of our CbO-family algorithms described in Chapter 2. Namely the performance efficient computation of formal concepts $\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\rangle$ with attribute y added to (the intent of) formal concept $\langle A, B\rangle$, the bitwise level representation of input data and intents of formal concepts and the (almost) overhead-free scalable parallelization of the computation of formal concepts (*Parallel CbO*).
A pseudocode of the modified *NextNeighbor* algorithm, which in fact would be the same as the pseudocode of the modified (recursive) *CbO* algorithm, can be found together with a description in [14].
The required formal concepts and our partial order relation on the concepts computed from the data table in Figure 3.2 (bottom) are depicted in Figure 3.3, by means of part of the concept lattice. Note that the cover relation on the concepts, displayed by solid lines in the figure, is a subset of our modified relation and the difference is displayed by dashed lines. The boldface solid lines indicate the tree to be selected from a collection of overlapping trees the concept lattice is considered as in our idea of the method (recall

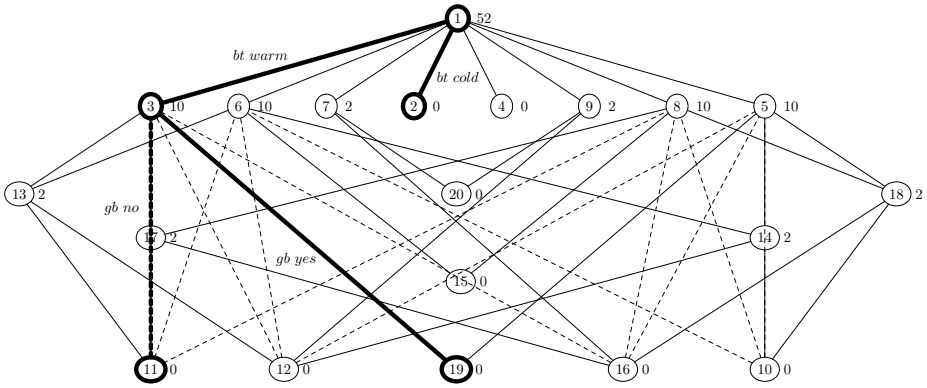

Figure 3.3: Part of the concept lattice and a tree of concepts (boldface solid lines) of data table in Figure 3.2.

the introduction Section 3.1.1). The tree is going to be selected using a procedure described in the following Step 2 of our decision tree induction method.

Step 2 – selecting a tree of formal concepts
In this step we select a tree from the partially ordered set of formal concepts computed in Step 1.
First, we calculate for each formal concept $\langle A, B\rangle$ computed in Step 1 the number $L_{\langle A,B\rangle}$ of all of its smaller related formal concepts $\langle (B \cup \{y_v\})^{\downarrow}, (B \cup \{y_v\})^{\downarrow\uparrow}\rangle$ in our partial order relation. Note that each such related concept is counted for each different attribute $y_v \in Y$ added to $\langle A, B\rangle$, cf. the rationale behind the relation above. The numbers $L_{\langle A,B\rangle}$ can be computed already together with computing the formal concepts and the relation in Step 1 (see the pseudocode in [14]).
Furthermore, for every formal concept $\langle A, B\rangle$ we define collections $\mathcal{N}^a_{\langle A,B\rangle}$ of formal concepts that are candidates to become the children of $\langle A, B\rangle$ in the selected tree. $\mathcal{N}^a_{\langle A,B\rangle}$ is the collection of smaller formal concepts $\langle (B \cup \{y_v\})^{\downarrow}, (B \cup \{y_v\})^{\downarrow\uparrow}\rangle$ related to $\langle A, B\rangle$ which result by adding a (binary) attribute y_v for every value v of (original) attribute a if the smaller concept was computed in Step 1; otherwise $\mathcal{N}^a_{\langle A,B\rangle}$ contains the least formal concept $\langle Y^{\downarrow}, Y\rangle$ in place of the smaller concept, and we put $L_{\langle Y^{\downarrow},Y\rangle} = \infty$.
Next, we select a tree from the partially ordered set of formal concepts computed in Step 1 by iteratively going from the largest formal concept to the minimal ones. The selection is based on the numbers $L_{\langle A,B\rangle}$ defined above.

(1) The root node of the tree is the largest formal concept $\langle X, X^{\uparrow}\rangle$.

(2) This step corresponds to selection of the splitting attribute. For every

formal concept $\langle A, B\rangle$ in the tree we construct we select from among all attributes of original input data the attribute a for which $\mathcal{N}^a_{\langle A,B\rangle}$ contains a formal concept $\langle C, D\rangle$ with the smallest number $L_{\langle C,D\rangle}$. The idea behind this rule is that a small value of $L_{\langle C,D\rangle}$ (the number of smaller formal concepts related to $\langle C, D\rangle$) indicates, in the optimistic scenario, a small number of subsequent decision steps in the resulted decision tree necessary to classify objects from A provided we start with a decision based on a, thus leading to a small decision tree (in order to provide a good generalization of the tree, cf. the decision trees preliminaries in Section 3.1.2).

In case of a tie, i.e. if $L_{\langle C_1,D_1\rangle} = L_{\langle C_2,D_2\rangle}$ for some $a_1 \neq a_2$ with $\langle C_1, D_1\rangle \in \mathcal{N}^{a_1}_{\langle A,B\rangle}$ and $\langle C_2, D_2\rangle \in \mathcal{N}^{a_2}_{\langle A,B\rangle}$, we select a_i for which the extent C_i is the largest (contains the largest number of objects). If there is still a tie, we break it arbitrarily. The selected attribute a is later used as the splitting attribute for the node of the resulted decision tree that corresponds to formal concept $\langle A, B\rangle$.

(3) Finally, for every formal concept $\langle A, B\rangle$ in the tree and the attribute a selected for $\langle A, B\rangle$ in (2) we connect $\langle A, B\rangle$ to each formal concept $\langle C, D\rangle$ from $\mathcal{N}^a_{\langle A,B\rangle}$ by an edge labeled by a binary attribute y for which $D = (B \cup \{y\})^{\downarrow\uparrow}$.

Again, a pseudocode of the just described algorithm that selects a tree from the partially ordered set of formal concepts can be found in [14]. One can find there also a step-by-step illustration of the description of the algorithm on the partially ordered set of formal concepts (a part of the concept lattice) presented in Figure 3.3. As noted above, the resulting selected tree of formal concepts is depicted in Figure 3.3 by the boldface solid lines.

Step 3 – converting the tree of formal concept into a decision tree
The last step of our decision tree induction method is the conversion of the tree of formal concepts into a decision tree. This step is straightforward. We take the tree obtained in Step 2 and re-label its nodes and edges. An inner node is labeled by the attribute (of original input data) selected in (2) of Step 2 for this node. For example, when constructing a decision tree from the tree depicted in Figure 3.3, the node corresponding to formal concept No. 3 is labeled by *gives birth*. An edge going from a node is labeled by the value of the attribute (of original input data) corresponding to the binary attribute used as a label of this edge in (3) of Step 2. For example, the edge labeled by *gb no* in Figure 3.3 is labeled by *no* in the resulting decision tree. The last problem is labeling of leaf nodes. For a leaf node corresponding to formal concept $\langle A, B\rangle$, the node is labeled by the class label which is the class label of all, or the predefined majority of, objects from A. If a leaf node n corresponds to the least formal concept $\langle Y^{\downarrow}, Y\rangle$ (which usually

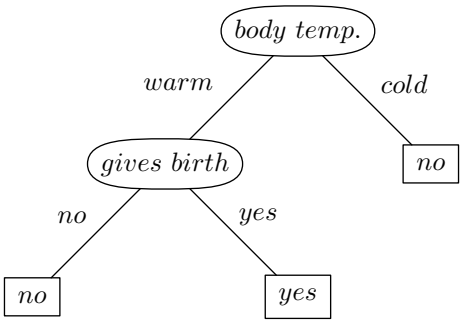

Figure 3.4: The decision tree induced from input data in Figure 3.2.

covers no objects), the node is labeled by the class label which would have been assigned to the parent node of n as if it was a leaf node.
The resulting decision tree induced from the input data in Figure 3.2 (top) which results by the conversion of the tree of formal concepts depicted in Figure 3.3 is depicted in Figure 3.4.

3.1.4 Experimental evaluation

To illustrate the classification performance of the presented decision tree induction method we include selected results from the experimental evaluation and comparison of the method to reference decision tree induction and other machine learning algorithms. The results were borrowed from [14], where one can find more.

Before going to the evaluation, let us first note in this context that the algorithm of our method is computationally more demanding than algorithms of other existing decision tree induction and machine learning methods (including those compared)—due to computing a possibly large number of formal concepts. The overall asymptotic worst-case time complexity of the method is thus given by the (partial order of) formal concepts computing step (Step 1 in the description of the method in Section 3.1.3), i.e. $O(|X||Y|^2|L|)$, where $|X|$ is the number of input data objects, $|Y|$ is the number of binary attributes after transformation from original input data attributes and $|L|$ is the number of computed formal concepts. However, for decision tree induction, and classification algorithms in general, classification performance, most typically given in terms of *classification accuracy*, i.e. the percentage of correctly classified objects from both training and testing data sets, is more important than induction time performance.
So, in short, we evaluated and compared our algorithm with decision tree induction algorithms *ID3* and *C4.5* [120] (entropy, or more precisely, information gain based), an instance based learning method (k-nearest neighbor clustering, for $k = 1$ further denoted *IB1*), and a multilayer perceptron neu-

Table 3.1: Classification accuracy for selected datasets (best results are in boldface).

training % testing %	"FCA based"	*ID3*	*C4.5*	*IB1*	*MLP*
breast-cancer	**88.631**	88.630	86.328	84.887	88.550
	79.560	75.945	79.181	71.901	**79.939**
kr-vs-kp	84.395	**84.674**	82.124	79.132	84.426
	74.656	74.503	72.780	68.886	**74.880**
mushroom	96.268	**97.517**	97.163	96.556	97.234
	96.284	96.602	**96.671**	95.214	95.992
spect	**92.250**	**92.250**	89.250	88.250	91.500
	55.187	54.866	59.679	59.251	**60.481**
tic-tac-toe	98.991	**100.000**	95.165	**100.000**	**100.000**
	85.197	80.519	78.539	83.262	**97.827**
vote	**97.528**	**97.528**	94.883	97.020	95.545
	90.507	89.280	86.500	**91.303**	88.106
zoo	**98.019**	**98.019**	96.039	97.799	97.678
	96.036	95.036	92.690	94.463	95.536
average	93.726	**94.088**	91.565	91.949	93.562
	82.490	80.964	80.863	80.611	**84.680**

ral network trained by back propagation [98] (*MLP*) [1] on selected public real-world datasets from the UCI Machine Learning Repository [102]. The datasets are from various areas like medicine, biology, games, politics or astronomy, basic characteristics of the datasets (numbers of objects, original and transformed binary attributes and class labels distribution) can be found in [14]. The selected results from experiments done using the 10-fold stratified cross-validation test [70] are depicted in Table 3.1. The table shows average percentage rates of correct classifications for both training (upper number in the table cell) and testing (lower number) data sets for each algorithm and dataset being compared, plus the average over all datasets. Boldface numbers denote the best results. Our method is called "FCA based" in the table.

We can see that our decision tree induction method outperformed *C4.5* and

[1] The algorithms were borrowed and run from Weka [137] (Waikato Environment for Knowledge Analysis, `http://www.cs.waikato.ac.nz/ml/weka/`), a software package that contains implementations of machine learning and data mining algorithms in Java. Default Weka's parameters were used for the algorithms.

IB1 and gains almost identical results to *ID3* and *MLP* on training data sets of all datasets. On the testing data sets, which is more important from the point of view of the evaluation, this is also the case with some exceptions for which *MLP* outperformed all of the compared methods. We refer to [14] for a more thorough discussion of the evaluation and comparison, also with further datasets. Anyway, the obtained results are very promising, it seems that our method outperforms instance based (k-nearest neighbor clustering) learning methods (*IB1*) and that it is able to provide better results than traditional decision tree induction, entropy based, methods (*ID3*, *C4.5*) and even neural network methods (*MLP*) on clear dense data. However, we are fully aware that more experiments on more datasets, with further decision tree induction and machine learning algorithms and methods, and also using more informative classification measures than accuracy taking into account also incorrect classifications (like *F-measure*, for instance), are needed to approve those conclusions.

3.1.5 Summary and topics for future research

We have presented a novel method of decision tree induction based on formal concept analysis. The method implements a straightforward idea of utilizing certain formal concepts of input data as nodes of the decision tree constructed from the data. The problem of selection of the formal concepts, which determines the selection of splitting attributes of the tree, is resolved by a heuristic based on the numbers of smaller formal concepts in a particularly defined partial order relation on all formal concepts of the input data. The intuition behind the method is to look at a part of the concept lattice of input data as a collection of overlapping trees and select one of those trees as the decision tree. To compute formal concepts (and the partial order) we can use some of the modified *CbO* algorithms described in Sections 2.2.1 to 2.2.3. The experimental evaluation and comparison to standard decision tree induction and machine learning methods indicates good classification performance. According to selected results in Section 3.1.4, our method outperforms an instance based learning method (k-nearest neighbor clustering) and is comparable to entropy-based decision tree induction algorithms *ID3* and *C4.5*.

The main novelty of our method, compared to existing approaches utilizing FCA in classification and machine learning, is in the utilization of the closure properties of formal concepts and the relationships between the concepts (in terms of the partial order relation on the concepts) directly in the process of construction of the decision tree.

At present state, the method requires, for the selection of splitting attributes of the decision tree, to compute a possibly large number of formal concepts and the partial order relation on the concepts while only a smaller number of computed concepts is subsequently selected to form the induced decision

tree. This can be seen as a bottleneck of the method (also from the point of view of time performance). But, on the other hand, once one already has the partial order set of the formal concepts computed (which is a part of the concept lattice), the selection of those concepts forming the decision tree is fast. This draws a possible perspective on using the method: decision tree induction and classification from already available concept lattices. The advantage over other methods would be the conceptual information hidden in the tree nodes (which are in fact formal concepts). Such information is not (directly) available by other methods.
There is obviously a lot of topics for future research, see [14] for more than the following selected:

- theoretical research on the relationships between formal concepts and decision tree nodes and a (partial order) relation on concepts and a decision tree itself, with en emphasis to the problem of selection of splitting attributes to explain the results of experiments and to further interpret the decision tree regarding the conceptual information hidden in its nodes,

- possibility to compute a smaller number of formal concepts from which the concepts constituting nodes of the decision tree are selected, i.e. to predict the number of smaller concepts to a given formal concept used in the selection,

- incremental update of the induced decision tree via incremental update algorithms of computing formal concepts or concept lattice of data which grows (object by object),

- more experiments on more datasets, with further decision tree induction and machine learning algorithms and methods, to approve conclusions from experimental evaluation and comparison in Section 3.1.4,

- dealing with incomplete and *noisy data*, i.e. data having missing or wrong values of some attributes for some objects or having conflicting class labels assigned to objects (sharing the same values of all attributes),

- tackling the problem of overfitting of the induced decision tree to a training data set—a common solution used is pruning the tree [120, 121, 122, 129], which means omitting some parts of the tree.

The decision tree induction method described in the above sections has been presented at conferences devoted to FCA and machine learning, *CLA 2007* and *EMCSR 2008*, with publications in the conference proceedings. The extended version of the respective papers has been published in the *Int. Journal of General Systems*.

3.2 Feature extraction using Boolean matrix factorization by means of FCA

3.2.1 Introduction

The second presented application of formal concepts and FCA concerns feature extraction (construction) problem where the application is through Boolean matrix factorization approached by means of FCA.

When applying a data mining or machine learning method, input data is often subject to some sort of preprocessing before the data is processed by the method. Usually to "help" the particular method to achieve better results [36, 43, 120, 130]. The quality of results provided by the methods heavily depends on the quality of input data description. In case of object-attribute relational data objects are described by attributes. Clearly, better attributes describing the objects lead to better results from a data mining and machine learning method. The general aim in input *data preprocessing* is to create, or extract, from the data (more precisely from various relationships, dependencies or hidden patterns in the data) new attributes that extend or even substitute the set of original attributes. The new attributes should better describe the objects in data than the less descriptive original attributes. Usually there is a less number of the new attributes than the original ones, which means a reduction of dimensionality of data. Here, a natural question arises, whether the reduced number of new attributes can better describe the input data or not. The methods of extraction or construction of the new attributes, called features in this area, are called *feature extraction*, or *feature construction*, methods [89, 57, 90].
Formal concept analysis (FCA) has often been proposed to be used for input data preprocessing [133, 97] but, interestingly, never as a feature extraction method. In this application, FCA can be utilized in a way that certain formal concepts are used to define new attributes which then describe objects in place of the original attributes. A key point is (again, cf. the Section 3.1 on using formal concepts as decision tree nodes) in the selection of the concepts. In the method which we are going to present below the selected formal concepts are concepts that correspond to so-called (Boolean) *factors* produced by a recently proposed method of *Boolean matrix factorization* based on FCA [27, 28]. This is a novel approach in which the factors can be considered as particular conjunctions of original attributes. The factors themselves are used as new attributes to describe objects, either extending the set of original attributes or substituting the original attributes. The latter usually means the reduction of dimensionality of data since the number of factors is usually smaller than the number of the original attributes [8]. Our method is briefly described in Section 3.2.3, the full description can be found in [108, 110].

A data mining or machine learning method which is very often used in the literature for existing feature extraction/construction methods to demonstrate and evaluate the methods is *decision tree induction*. Hence we followed this choice and, in fact, adapt the Boolean matrix factorization for the factors to play the role of new attributes right in the decision tree induction. In an experimental evaluation of our method, where we compared classification performance of the reference decision tree induction methods *ID3* and *C4.5* on the original and preprocessed public real-world benchmark datasets, we obtained good results of the method in that the performance was better for the preprocessed data than for the original data. See Section 3.2.4 for a selection of the results which are summarized in [108, 110].

In the literature, the most relevant to our method are methods known as *constructive induction* [94, 128]. Here, new compound attributes are constructed from original attributes as logical conjunctions and/or disjunctions of the attributes [112] or as combinations using arithmetic operations [114], or the new attributes are expressed in the m-of-n form [99]. Considering decision trees as the target data mining or machine learning method after preprocessing, *oblique decision trees* [60, 101] are also connected to our approach—in a sense that multiple original attributes are used as a compound splitting attribute (see Section 3.1.2 for the notion of splitting attribute) instead of a single attribute at a time. Typically, linear combinations of attributes are sought, see e.g. [32, 132]. However, in comparison to evaluating a single attribute to be good splitting attribute, it is quite computationally challenging to find and evaluate groups of original attributes to form a good compound splitting attribute.

Regarding FCA and concept lattices, there have been several FCA-based approaches on construction of a whole learning (most often classification) model, e.g. [82] or [92], see [47] for a survey and comparison. They are commonly called *lattice-based* or *concept-based machine learning approaches* [45, 113] (cf. also Section 3.1.1). But, as mentioned above, the usage of FCA to create or extract from input data some new (compound) attributes better describing the objects is discussed very marginally or not at all in existing papers.

3.2.2 Preliminaries in Boolean matrix factorization in terms of FCA

In order to describe our method of feature extraction we first have to briefly introduce basics of Boolean matrix factorization problem, on which the method is based, and how it is solved by means of FCA. Necessary are also transformations between (original) attribute and factor (new attribute) spaces, described in Section 3.2.2, to be able to describe by factors objects originally described by attributes and vice-versa.

Boolean Matrix Factorization (BMF), also referred to as *Boolean Factor Analysis (BFA)* or factor analysis of binary (Boolean) data, is a Boolean (binary) matrix decomposition method which provides a representation of an object-attribute binary data matrix (a matrix with entries 0 or 1) by a Boolean product of two different binary matrices, one describing objects by new attributes called *factors*, and the other describing factors by the original attributes [59, 65].
Stated as a problem, the aim of BMF is to find a decomposition

$$I = A \circ B \tag{3.1}$$

of a $n \times m$ binary matrix I into a Boolean product $A \circ B$ of an $n \times k$ binary matrix A and a $k \times m$ binary matrix B with k as small as possible. Thus, instead of m original attributes, one aims to find k new attributes, called *factors*. A Boolean product $A \circ B$ of binary matrices A and B is defined by

$$(A \circ B)_{ij} = \max_{l=1}^{k} \min(A_{il}, B_{lj}).$$

The inner dimension, k, in the product may be interpreted as the number of factors that are used as new attributes to describe the original data. Namely, $A_{il} = 1$ means that factor l applies to object i and $B_{lj} = 1$ means that attribute j is one of the manifestations of factor l. The factor model behind ((3.1)) has therefore the following meaning: The object i has the attribute j if and only if there exists a factor l that applies to i and j is one of the manifestations of l. As an example,

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

We refer to [27] for further information and references to papers that deal with the problem of factor analysis and decompositions of binary matrices. Recently, a solution to the problem of finding the decomposition (3.1) with the number k of factors as small as possible was described in [27, 28] by means of formal concept analysis. The description lies in an observation that matrices A and B can be constructed from a set $\mathcal{F}$ of formal concepts of matrix I, considered as formal context $\langle X, Y, I \rangle$ (see Section 1.1), where $X = \{1, \ldots, n\}$, $Y = \{1, \ldots, m\}$ (objects and attributes of the context correspond to the rows and columns of I) and binary relation I of the context corresponds to matrix I in an obvious way. In particular, let

$$\mathcal{F} = \{\langle C_1, D_1 \rangle, \ldots, \langle C_k, D_k \rangle\} \tag{3.2}$$

be a set of formal concepts of $\langle X, Y, I \rangle$, a subset of the set $\mathcal{B}(X, Y, I)$ of all

formal concepts of $\langle X, Y, I\rangle$. Consider the $n \times k$ binary matrix $A_{\mathcal{F}}$ and a $k \times m$ binary matrix $B_{\mathcal{F}}$ defined by

$$(A_{\mathcal{F}})_{il} = 1 \text{ iff } i \in C_l \quad \text{and} \quad (B_{\mathcal{F}})_{lj} = 1 \text{ iff } j \in D_l, \tag{3.3}$$

i.e. the l-th column $(A_{\mathcal{F}})_{\_l}$ of $A_{\mathcal{F}}$ consists of the characteristic vector of A_l and the l-th row $(B_{\mathcal{F}})_{l\_}$ of $B_{\mathcal{F}}$ consists of the characteristic vector of B_l. Denote by $\rho(I)$ the smallest number k, so-called *Schein rank* of I, such that a decomposition of I exists with k factors. The following theorem shows that using formal concepts as in ((3.3)) enables us to reach the Schein rank, i.e. is in this sense optimal:

Theorem 3 ([27]) *For every binary matrix I, there exists $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ such that $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$ and $|\mathcal{F}| = \rho(I)$.*

Formal concepts $\mathcal{F}$ in the theorem are called *factor concepts*. Each factor concept determines a factor. For a constructive proof of the theorem we refer to [27]. As it has also been demonstrated in [27], a useful feature of using formal concepts for determining factors is the fact that formal concepts may be easily interpreted. Namely, every factor, by means of a formal concept $\langle C_l, D_l\rangle$, consists of a set C_l of objects (formal concept extent) the factor applies to, a set D_l of attributes (formal concept intent) which are manifestations of the factor and C_l contains just the objects to which all the attributes from D_l apply and D_l contains just all attributes shared by all objects from C_l (cf. the closure property of formal concepts in Section 1.1). The factors thus have a natural, easy to understand meaning.
Note that the problem of finding the set of factors (factor concepts of $\langle X, Y, I\rangle$) the size of which equals the Schein rank of I is NP-hard (which can be shown e.g. by reduction to the set covering optimization problem). Due to that a *greedy approximation algorithm* for finding the factors was proposed in [27], denoted as *Algorithm 2* there. This algorithm is used for finding factors (factor concepts) in our method of feature extraction introduced in [110] and summarized, after the following necessary note, in Section 3.2.3.

Transformations between attribute and factor spaces

For an object we can consider its representations in the m-dimensional Boolean space $\{0,1\}^m$ of (original) attributes and in the k-dimensional Boolean space $\{0,1\}^k$ of factors. For an object-attribute matrix I and an object-factor matrix A, in the space of attributes, the vector representing object i is the i-th row of I, and in the space of factors, the vector representing i is the i-th row of A.
Natural transformations between the space of attributes and the space of factors is described by the mappings $g\colon \{0,1\}^m \to \{0,1\}^k$ and $h\colon \{0,1\}^k \to$

$\{0,1\}^m$ defined for $P \in \{0,1\}^m$ and $Q \in \{0,1\}^k$ by

$$(g(P))_l = \min_{j=1}^{m}(B_{lj} \to P_j), \tag{3.4}$$

$$(h(Q))_j = \max_{l=1}^{k} \min(Q_l, B_{lj}), \tag{3.5}$$

for $1 \leq l \leq k$ and $1 \leq j \leq m$. Here, $\to$ denotes the truth function of classical implication logical operation ($1 \to 0 = 0$, otherwise 1). (3.4) says that the l-th component of $g(P) \in \{0,1\}^k$ is 1 if and only if for every attribute j, $P_j = 1$ for all positions j for which $B_{lj} = 1$, i.e. the l-th row of B is included in P. (3.5) says that the j-th component of $h(Q) \in \{0,1\}^m$ is 1 if and only if there is factor l such that $Q_l = 1$ and $B_{lj} = 1$, i.e. attribute j is a manifestation of at least one factor from Q.
And, if the decomposition $I = A \circ B$ uses formal concepts for determining factors, we have:

Theorem 4 ([27]) *For $i \in \{1, \ldots, n\}$,*

$$g(I_{i\_}) = A_{i\_} \quad \textit{and} \quad h(A_{i\_}) = I_{i\_}.$$

That is, g maps the rows of I to the rows of A and vice versa, h maps the rows of A to the rows of I.

For other results showing properties and describing the geometry behind the mappings g and h, see [27].

3.2.3 Boolean factors as new attributes

We can now summarize our feature extraction method which utilizes Boolean matrix factorization based on FCA.

Note first that, as indicated in the introduction section 3.2.1, the machine learning method which we will use to demonstrate and evaluate the method is decision tree induction and in classification, input data attributes of various types are used (often categorical, as previously noted in Section 3.1.3). So, in order to utilize Boolean matrix factorization (BMF, and FCA as a matter), we again need to apply a transformation of such attributes to binary attributes. The transformation applied is again the conceptual scaling [51] and as well we need not transform the class labels assigned to objects because we deal with attributes (only) in the feature extraction preprocessing of data. BMF, by means of FCA, which we use in our method is applied on data which we obtain after such transformation.
The approach utilized in our feature extraction method consists in using as new (additional or substituting) attributes Boolean factors obtained by Boolean matrix factorization based on FCA. Hence the Boolean (binary)

matrix decomposition of input data table is performed as a key part of the method. As noted in Section 3.2.2, the algorithm which we use (in [108, 110]) for the decomposition is the greedy approximation algorithm from [27], denoted as *Algorithm 2* there, which computes factors as formal concepts (factor concepts). However, the *criterion of optimality* of computed factors (factor concepts) utilized in the greedy heuristic search for factors in the algorithm is modified in our application. In short, the algorithm (and other binary matrix decomposition algorithms often too) applies a greedy heuristic approach to search in the space of all formal concepts for concepts which cover the largest area of still uncovered 1s in the input data table. The criterion function of optimality of a factor is thus the "cover ability" of the factor concept determining the factor, in particular the number of uncovered 1s in the input data table which are covered by the concept, see [27]. For further use, we translate the function value to interval $[0, 1]$ (with the value of 1 meaning the most optimal) by dividing the value by the total number of still uncovered 1s in the input data table. However, since we use the factors, as new attributes, to aid a machine learning method, the decision tree induction in particular, we additionally attempt to look for factors which also have good "decision ability", i.e. that the factors are good candidates to be splitting attributes in a decision tree. Thus, in the modified decomposition algorithm, we use a combination of the two criteria into a combined criterion function of optimality of factors (factor concepts).

Let $I \subseteq X \times Y$ be our input data table describing objects $X = \{1, \ldots, n\}$ by binary attributes $Y = \{1, \ldots, m\}$. The combined criterion function $c\colon 2^{X \times Y} \to [0, 1]$ of optimality of factor concept $\langle A, B\rangle$ (of $\langle X, Y, I\rangle$) is defined as:

$$c(\langle A, B\rangle) = w \cdot c_A(\langle A, B\rangle) + (1 - w) \cdot c_B(\langle A, B\rangle), \tag{3.6}$$

where $c_A(\langle A, B\rangle) \in [0, 1]$ is the (original) criterion function of "cover ability" of factor concept $\langle A, B\rangle$, $c_B(\langle A, B\rangle) \in [0, 1]$ is a new criterion function of "decision ability" of factor concept $\langle A, B\rangle$ and w is the weight of preference (given by user) among the functions c_A and c_B. Now, we have introduced above the function c_B as to be a measure of merit of a factor, determined by a factor concept, as a splitting attribute. In decision trees, common approaches to the selection of splitting attribute are based on entropy measures, as mentioned in Section 3.1.2 on decision tree preliminaries. Basically, in those approaches, an attribute is the better splitting attribute for a current collection of objects the lower is the weighted sum of entropies of sub-collections of objects obtained after splitting the current collection of objects based on the attribute. We thus design the function c_B to resemble

$$\begin{pmatrix} 0&1&0&1&0&1&1&0\\ 0&1&0&1&1&0&0&1\\ 1&0&1&0&0&1&0&1\\ 0&1&1&0&1&0&1&0\\ 1&0&0&1&1&0&1&0 \end{pmatrix} = \begin{pmatrix} 0&0&1&0&0&1\\ 0&0&1&0&1&0\\ 0&0&0&1&0&0\\ 1&0&0&0&0&0\\ 0&1&0&0&0&0 \end{pmatrix} \circ \begin{pmatrix} 0&1&1&0&1&0&1&0\\ 1&0&0&1&1&0&1&0\\ 0&1&0&1&0&0&0&0\\ 1&0&1&0&0&1&0&1\\ 0&1&0&1&1&0&0&1\\ 0&1&0&1&0&1&1&0 \end{pmatrix}$$

Figure 3.5: Boolean matrix decomposition of example input data from Figure 3.2.

this:

$$c_B(\langle A, B\rangle) = 1 - \left(\frac{|A|}{|X|} \cdot \frac{E(\text{class}|A)}{-\log_2 \frac{1}{|V(\text{class}|A)|}} + \frac{|X \setminus A|}{|X|} \cdot \frac{E(\text{class}|X \setminus A)}{-\log_2 \frac{1}{|V(\text{class}|X \setminus A)|}} \right), \tag{3.7}$$

where $V(\text{class}|A)$ is the set of class labels assigned to objects A and $E(\text{class}|A)$ is an entropy measure of objects A over the class labels. As an entropy measure, we use the classical Shannon's entropy:

$$E(\text{class}|A) = -\sum_{l \in V(\text{class}|A)} p(l|A) \cdot \log_2 p(l|A), \tag{3.8}$$

where $p(l|A)$ is the fraction of objects A which are assigned with class label l. Note that the formula $-\log_2 \frac{1}{|V(\text{class}|A)|}$ in (3.7) represents the maximal possible value of (Shannon's) entropy of objects A in the case the class labels $V(\text{class}|A)$ are assigned to the objects evenly and the purpose of it is to normalize the value of c_B to interval $[0, 1]$. Note also that we consider $\frac{0}{0} = 0$ in calculations in 3.7.

To illustrate our BMF-based feature extraction method, let us consider I as a $n \times m$ binary matrix and find a decomposition $I = A \circ B$ of I into the $n \times k$ matrix A describing objects by factors $F = \{f_1, \ldots, f_k\}$ and $k \times m$ matrix B explaining factors F by attributes. The decomposition of the example data in Figure 3.2 (top, page 54, introduced in Section 3.1.3 on the decision tree induction method via formal concepts), for the new criterion function of optimality of factors (factor concepts) and any value of the weight w discussed above, is depicted in Figure 3.5.

When extending the set Y of original attributes with factors F as new attributes, the set Y' of attributes of the extended data is $Y \cup F$ and the extended data table $I' \subseteq X \times Y'$ is the apposition of the original data table and the table representing the matrix A describing objects by factors, i.e. $I' \cap (X \times Y) = I$ and $I' \cap (X \times F) = A$. The extended data table for our example data is illustrated in Figure 3.6.

When substituting the set Y of original attributes by factors F as new

animal	*bc*	*bw*	*gn*	*gy*	*fn*	*fy*	*hn*	*hy*	f_1	f_2	f_3	f_4	f_5	f_6	*mammal*
cat	0	1	0	1	0	1	1	0	0	0	1	0	0	1	yes
bat	0	1	0	1	1	0	0	1	0	0	1	0	1	0	yes
salamander	1	0	1	0	0	1	0	1	0	0	0	1	0	0	no
eagle	0	1	1	0	1	0	1	0	1	0	0	0	0	0	no
guppy	1	0	0	1	1	0	1	0	0	1	0	0	0	0	no

Figure 3.6: Extended data table for example input data from Figure 3.2.

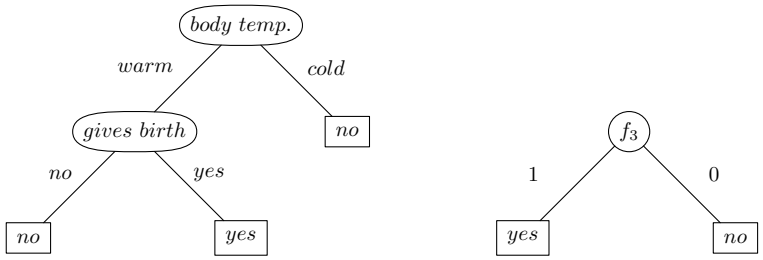

Figure 3.7: Decision trees induced from original data table in Figure 3.2 (left) and from the data table in Table 3.6 (right), either full or restricted to factors only.

attributes, the set Y' of attributes of the substituted data is the set F of factors and the substituted data table $I' \subseteq X \times Y'$ is the table representing the matrix A describing objects by factors. Obviously, this data table for our example data in Figure 3.2 is the table illustrated in Figure 3.6 restricted to the factors $f_1, \ldots, f_6$.

Now, to evaluate the factors as new attributes, a decision tree is induced from the new (extended or substituted) data table $I' \subseteq X \times Y'$ instead of the original data table I. The class labels assigned to objects in the new data table remain unchanged in the case of the extended data table, in the case of the substituted data table see a note below. In our example, a decision tree induced from the data table in Figure 3.6 (either full or restricted to the factors $f_1, \ldots, f_6$, the induced trees are the same) is depicted in Figure 3.7 (right). For the purpose of comparison, a decision tree induced from the original data table from Figure 3.2 is depicted in Figure 3.7 (left). We can see that objects from the new data table containing (also) factors as new attributes can be classified by a single attribute, namely, the factor f_3. The manifestations of the factor are the original attributes *bt warm* and *gb yes*, hence, the factor f_3, as a particular conjunction of the two attributes, is better splitting attribute in decision tree induction from the data than the two original attributes alone.

When classifying an object x described by (original) attributes Y as a vector $P_x \in \{0,1\}^m$ in the space of attributes, we need first to transform the description of x to the space of factors F as a vector $g(P_x) \in \{0,1\}^k$ by

mapping (3.4). In the mapping, matrix B explaining factors by original attributes is used. Then the object described by concatenation of P_x and $g(P_x)$ (in the case of extending the set of attributes by factors) or by $g(P_x)$ alone (in the case of substituting attributes by factors) is classified by the decision tree in a usual way.
Note that in the case of substituting the original attributes by factors we need to resolve the following important issue. Since the number of factors is usually smaller than the number of original attributes (and the substitution then leads to the reduction of dimensionality of data as mentioned in the introduction section 3.2.1), the transformation of input data objects from the attribute space to the factor space described in Section 3.2.2 is not an injective mapping. Namely, for two distinct objects $x_1, x_2 \in X$ with different attribute values, i.e. described by different vectors in the space of attributes, $P_{x_1} \neq P_{x_2}$, which have different class labels assigned, $\text{class}(x_1) \neq \text{class}(x_2)$, the representation of both x_1, x_2 by vectors in the factor space might be the same, $g(P_{x_1}) = g(P_{x_2})$.
To cope with this situation, at present [108, 110] we adopt a common approach used in decision tree induction to avoid overfitting of data or to cope with noise and/or discrepancies in data—namely, a class label which we assign to each object $x \in X$ in the new data table I' (where objects are represented in factor space) is the majority class label of objects in the original data table I (represented in attribute space) which are mapped (by mapping (3.4)) to the (same) object x. I.e. we assign the class label which is assigned to the most of the objects.

3.2.4 Experimental evaluation

Here we briefly illustrate the impact of preprocessing the input data by our feature extraction method, by presenting selected results obtained from an experimental evaluation of the method borrowed from [110].

In the evaluation we compared classification performance of the reference decision tree induction methods *ID3* and *C4.5* [120] on original input data and on the same, preprocessed, data after substituting original attributes by factors computed using Boolean matrix factorization based on FCA as described in the previous Section 3.2.3. [2] The data we used are the selected public real-world datasets from the UCI Machine Learning Repository [102], from various areas like medicine, biology, games or politics; the basic characteristics of the datasets (numbers of objects, original and transformed binary attributes and class labels distribution) can be found in [110]. An illustrative sample of results from experiments done using the 10-fold stratified cross-validation test [70] are depicted in the tables in Table 3.2. The

[2] The algorithms were again borrowed and run from Weka [137], see the footnote at page 59.

Table 3.2: Classification accuracy increase for selected datasets, for factor selection set to "cover ability" (top table) and to "decision ability" (bottom table) of a factor concept.

	breast-cancer	kr-vs-kp	mushroom	tic-tac-toe	vote	average
ID3	+2.0 % +15.9 %	0 % −0.7 %	0 % 0 %	0 % +12.3 %	0 % −0.7 %	+0.4 % +5.36 %
C4.5	+3.1 % −1.1 %	−0.2 % −0.6 %	0 % 0 %	+2.8 % +9.2 %	−0.2 % −0.6 %	+1.1 % +1.38 %

	breast-cancer	kr-vs-kp	mushroom	tic-tac-toe	vote	average
ID3	+2 % +15.3 %	0 % 0 %	0 % 0 %	0 % +15.7 %	0 % +1.7 %	+0.4 % +6.54 %
C4.5	+4.7 % +3.5 %	0 % −0.2 %	0 % 0 %	+3.3 % +13.8 %	0 % +0.7 %	+1.6 % +3.56 %

tables show average increase in classification accuracy of decision trees induced from the preprocessed data (with original attributes substituted by factors) compared to decision trees induced from the original data, for both training (upper number in table cells) and testing (lower number) data sets for each algorithm and dataset being used, plus the average over all datasets. The top table shows the numbers for the case the criterion function of optimality of factors (see the description of the factor concept selection in Section 3.2.3) was set entirely to the criterion function of "cover ability" of a factor concept (c_A in the description), i.e. the original criterion of the used algorithm [27] for computing factors. This corresponds to setting $w = 1$ in the formula (3.6) of the combined criterion function. For the bottom table, we set $w = 0$ in (3.6), i.e. the criterion function of optimality was set entirely to our new criterion function of "decision ability" described (as c_B) in Section 3.2.3. This means that factors were selected just to be good splitting attributes in a decision tree based on an entropy measure.

We can clearly see that, while inducing at average slightly better (almost never worse) decision trees on training data sets, the decision tree induction methods induce at average significantly better decision trees, i.e. achieve higher classification accuracy, on testing data sets for datasets preprocessed by our feature extraction method where original attributes of a dataset were substituted by factors. For instance, *ID3* has better classification performance by 5.36 % for the criterion function of optimality of factors set to "cover ability" of the determining factor concepts, while for the criterion function of optimality set to our new criterion function of "decision ability" the performance is better by 6.54 %. More results from the evaluation with more datasets and machine learning methods other than decision trees can be found in [108, 110]. Let us only note that, according to our experiments, the results for extending the set of original attributes with factors

instead of substituting them are very similar, with ±1 % difference at average. This suggests that the decision tree induction from data with factors added as new additional attributes to the original attributes uses merely the factors as splitting attributes rather that the original attributes during the construction of a decision tree.

3.2.5 Summary and topics for future research

We have presented a novel feature extraction (construction) method applying Boolean matrix factorization (BMF) based on formal concept analysis [27]. In the method, new input data attributes (features) are extracted (constructed) as factors represented by selected formal concepts, which is the main novelty of the method. In the input data preprocessing usage, i.e. before the data are processed by some data mining or machine learning method, the factors, as new attributes, are used either to extend or substitute the set of original attributes. In the latter case it usually means reduction of dimensionality of the input data since the number of factors is usually smaller than the number of original attributes. For computing the formal concepts representing factors we use the algorithm from [27] in which the criterion of optimality of factors was additionally modified for the factors being used as new attributes for classification. The algorithm is also implemented using the single formal concept computation and data representation advantages described in Chapter 2.
As a data preprocessing step, the method was demonstrated on a classification problem represented by decision tree induction and experimental evaluation indicated usefulness of such preprocessing of data. Namely, decision trees induced from preprocessed data (with original attributes either extended or substituted by factors) outperformed decision trees induced from the original data, for two standard (entropy-based) decision tree induction methods *ID3* and *C4.5*. This is true especially when factors were selected in the BMF algorithm based on a particular entropy measure. Using the factors instead of, or in addition to, the original attributes thus leads to improving the classification performance.
Topics for future research include:

- better solving the issue of mapping distinct objects in original input data to the same object in preprocessed data created by substituting the original attributes by a smaller number of factors, as described in Section 3.2.3—the idea is to further modify the criterion function of optimality of factors in such a way that the resulting collection of factors is good if the number of such mapping cases is low,

- theoretical research on the role of factors as new attributes in the machine learning methods, particularly decision tree induction, to explain

the results of experiments, with a focus to design better criterion function of optimality of factors for the methods to improve their results; e.g. inspect and use more advanced (entropy-based) measures utilized in decision tree induction,

– evaluation of the usage of approximate matrix decomposition in BMF instead of exact; actually, the topic of utilizing approximate BMF (described in terms of FCA) for data dimensionality reduction from a data mining point of view is studied in [75],

– dynamic adjustment of the weight among the several criterion functions of optimality of factors combined to create the final function during factor computation (see Section 3.2.3) based on measuring and evaluating the factor optimality indicators ("cover ability", "decision ability", number of the mappings of distinct objects to the same object and other) – the idea is to suppress the negative indicators first and then boost the positive ones,

– evaluation of impact on the quality of classification of the various BMF methods from the literature, other than the used one from [27]; actually, some advancements in this topic may be found in our recent papers [19, 20],

– more experiments on more datasets, with further machine learning algorithms and methods, to justify the results of present experiments; there are more datasets and machine learning methods in experiments in [20].

The feature extraction method described in the above sections has been presented at conferences devoted to FCA and machine learning, *CLA 2010, ICMLA 2010* and *CLA 2012*, with publications in the conference proceedings. The extended version of the last of the respective papers has been published in the *Annals of Mathematics and Artificial Intelligence*.

Conclusion

This thesis presents selected results obtained by the author at the Department of Computer Science, Palacký University Olomouc, during years 2007–2012 (with remarks to further results from years 2013–2014), on the algorithms and applications of formal concept analysis (FCA)—a modern and intensively studied method for mining and analysis of object-attribute relational data, which enjoys an increasing interest and popularity in a growing number of communities.

Although FCA is nowadays a well-established and elaborated method with strong mathematical foundations, the current algorithms for computing formal concepts, the basic units of mined and analyzed data in the method, developed within FCA community are sufficient for middle-size data and their performance for large-scale data is not satisfactory. Therefore, in Chapter 2, we presented new performance efficient algorithms for computing formal concepts, which outperform almost all other known algorithms for computing formal concepts from the FCA literature, often by magnitudes, and allow to process large-scale data in a reasonable time. The algorithms are fully comparable, regarding the performance, with existing data mining algorithms.
During its development, FCA has also been applied in many fields. Among others, the usefulness of application of FCA has been demonstrated in the literature in classification and it is also very often used in data mining as a preprocessing method. In Chapter 3 we presented our contributions to applications of FCA in those two fields. First, by development of a novel decision tree induction method which utilizes formal concepts in the construction of decision tree and indicates good classification performance. Second, by utilizing formal concepts through Boolean matrix factorization based on FCA, in a novel feature extraction method capable of reducing the dimensionality of data and, evaluated on classification, improving classification results for preprocessed data over for the original data.

The presented research on the algorithms and applications of FCA is by no means finished. We have many topics for further development in both directions. Some were listed in the summaries of the corresponding chapters and

sections. Moreover, there are other directions of development and usage of FCA and related methods studied at the Department of Computer Science, Palacký University Olomouc, in which the author has also contributed in the past and which are being further developed.

References

[1] Agrawal R., Imielinski T., Swami A. N.: Mining association rules between sets of items in large databases. *Proc. ACM Int. Conf. of Management of Data* 1993, 207–216.

[2] Ammons G., Mandelin D., Bodik R., Larus J. R.: Debugging temporal specifications with concept analysis. *Proc. ACM SIGPLAN'03 Conference on Programming Language Design and Implementation*, 182–195.

[3] Andrews S.: In-Close, a Fast Algorithm for Computing Formal Concepts. In: Rudolph S., Dau F., Kuznetsov S. O. (Eds.): *Supplementary Proceedings of ICCS '09*, CEUR WS 483, 14 pp.

[4] Andrews S.: In-Close2, a High Performance Formal Concept Miner. *Proc. ICCS 2011*, *Lecture Notes in Computer Science* 6828, 50–62.

[5] Angiulli F., Cesario E., Pizzuti C.: Random walk biclustering for microarray data. *Information Sciences* 178(6)(2008), 1479–1497.

[6] Asuncion A., Newman D.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, 2007.

[7] Baklouti F., Levy G.: A distributed version of the Ganter algorithm for general Galois Lattices. In: Belohlavek R., Snasel V. (Eds.): *CLA 2005: Proceedings of the 3rd International Conference on Concept Lattices and Their Applications*, 207–221.

[8] Bartl E., Rezankova H., Sobisek L.: Comparison of Classical Dimensionality Reduction Methods with Novel Approach Based on Formal Concept Analysis. *Proc. RSKT 2011*, *Lecture Notes in Computer Science* 6954, 26–35.

[9] Bělohlávek R.: Fuzzy concepts and conceptual structures: induced similarities. *Proc. Joint Conf. Inf. Sci.'98*, Vol. I, 179–182.

[10] Belohlavek R.: *Fuzzy Relational Systems: Foundations and Principles.* Kluwer, Academic/Plenum Publishers, New York, 2002.

[11] Belohlavek R.: Lattices of fixed points of fuzzy Galois connections. *Math. Logic Quarterly* 47(1)(2001), 111–116.

[12] Belohlavek R.: What is a Fuzzy Concept Lattice? II. *Proc. RSFDGrC 2011*, *Lecture Notes in Artificial Intelligence* 6743, 19–26.

[13] Bělohlávek R., De Baets B., Outrata J., Vychodil V.: Inducing decision trees via concept lattices. In: Diatta J., Eklund P., Liquire M. (Eds.): *Proc. CLA 2007*, 274–285.

[14] Belohlavek R., De Baets B., Outrata J., Vychodil V.: Inducing decision trees via concept lattices. *Int. Journal of General Systems* 38(4)(2009), 455–467.

[15] Belohlavek R., De Baets B., Outrata J., Vychodil V.: Trees in Concept Lattices. In: Torra V., Narukawa Y.,Yoshida Y. (Eds.): Modeling Decisions for Artificial Intelligence: 4th International Conference, MDAI 2007, *Lecture Notes in Artificial Intelligence* 4617, 174–184.

[16] Belohlavek R., De Baets B., Outrata J., Vychodil V.: Characterizing trees in concept lattices. *Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 16(1)(2007), 1–15.

[17] Bělohlávek R., Funioková T., Vychodil V.: Galois connections with hedges. In: Liu Y., Chen G., Ying M. (Eds.): *Fuzzy Logic, Soft Computing & Computational Intelligence: Eleventh International Fuzzy Systems Association World Congress*, Vol. II, 2005, 1250–1255.

[18] Belohlavek R., Kostak M., Osicka P.: Formal concept analysis with background knowledge: a case study in paleobiological taxonomy of belemnites. *Int. Journal of General Systems* 42(4)(2013), 426–440.

[19] Belohlavek R., Outrata J., Trnecka M.: Impact of Boolean factorization as preprocessing methods for classification of Boolean data. In: Szathmary L., Priss U. (Eds.): *CLA 2012: Proceedings of the 9th International Conference on Concept Lattices and Their Applications*, 305–316.

[20] Belohlavek R., Outrata J., Trnecka M.: Impact of Boolean factorization as preprocessing methods for classification of Boolean data. *Annals of Mathematics and Artificial Intelligence* 72(12)(2014), 3–22.

[21] Bělohlávek R., Outrata J., Vychodil V.: Thresholds and shifted attributes in formal concept analysis of data with fuzzy attributes. In: Schärfe H., Hitzler P., hrstrøm P. (Eds.): *Proc. ICCS 2006*, *Lecture Notes in Artificial Intelligence* 4068, 117–130.

[22] Belohlavek R., Sigmund E., Zacpal J.: Evaluation of IPAQ questionnaires supported by formal concept analysis. *Information Sciences* 181(2011), 1774–1786.

[23] Bělohlávek R., Sklenář V., Zacpal J.: Crisply Generated Fuzzy Concepts. In: Ganter B., Godin R. (Eds.): *Proc ICFCA 2005*, *Lecture Notes in Computer Science* 3403, 268–283.

[24] Bělohlávek R., Sklenář V., Zacpal J., Sigmund E.: Evaluation of questionnaires supported by formal concept analysis. *Proc. CLA 2007*, 96–108.

[25] Belohlavek R., Trnecka M.: Basic Level in Formal Concept Analysis: Interesting Concepts and Psychological Ramifications. *Proc. IJCAI 2013*, 1233–1239.

[26] Belohlavek R., Trnecka M.: Basic level of concepts in formal concept analysis. *Proc. ICFCA 2012*, *Lecture Notes in Computer Science* 7278, 28–44.

[27] Belohlavek R., Vychodil V.: Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. System Sci.* 76(1)(2010), 3–20.

[28] Belohlavek R., Vychodil V.: Factor analysis of incidence data via novel decomposition of matrices. *Lecture Notes in Artificial Intelligence* 5548(2009), 83–97.

[29] Belohlavek R., Vychodil V.: On boolean factor analysis with formal concept as factors. *Proceedings of SCIS & ISIS 2006,*, 1054–1059.

[30] Bělohlávek R., Vychodil V.: What is a fuzzy concept lattice? *Proc. CLA 2005*, 34–45.

[31] Berry A., Bordat J. P., Sigayret A.: A local approach to concept generation. *Annals of Mathematics and Artificial Intelligence*, 49(2007), 117–136.

[32] Breiman L., Friedman J. H., Olshen R., Stone C. J.: *Classification and Regression Trees.* Chapman & Hall, NY, 1984.

[33] Burusco A., Fuentes-Gonzáles R.: The study of the L-fuzzy concept lattice. *Mathware & Soft Computing*, 3(1994), 209–218.

[34] Carpineto C., Romano G.: A Lattice Conceptual Clustering System and Its Aplication to Browsing Retrieval. *Machine Learning* 24(1996), 95–122.

[35] Carpineto C., Romano G.: *Concept Data Analysis. Theory and Applications.* J. Wiley, 2004.

[36] Cherkassky V., Mulier F.: *Learning from Data: Concepts, Theory, and Methods.* Wiley Interscience, 1998.

[37] Cole R., Eklund P., Stumme G.: Document Retrieval for Email Search and Discovery using Formal Concept Analysis. *Applied Artificial Intelligence* 17(3)(2003), 1–28.

[38] Correia J. H., Stumme G., Wille R., Wille U.: Conceptual knowledge discovery—a human-centered approach. *Applied Artificial Intelligence* 17(3)(2003), 281–302.

[39] Dau F., Ducrou J., Eklund P.: Concept Similarity and Related Categories in SearchSleuth. *Proc. ICCS 2008*, *Lecture Notes in Artificial Intelligence* 5113, 255–268.

[40] Dean J., Ghemawat S.: MapReduce: simplified data processing on large clusters. *Commun. ACM* 51(1)(2008), 107–113.

[41] Dekel U., Gill Y.: Visualizing class interfaces with formal concept analysis. *OOPSLA'03*, 288–289.

[42] Ducrou J., Eklund P.: An Intelligent User Interface for Browsing and Search MPEG-7 Images using Concept Lattices. *Int. Journal of Foundations of Computer Science* 19(2)(2008), 359–381.

[43] Dunham M. H.: *Data Mining. Introductory and Advanced Topics.* Prentice Hall, Upper Saddle River, NJ, 2003.

[44] Everitt B. S., Landau S., Leese M.: *Cluster Analysis (4th Ed.).* Oxford University Press, New York, 2001.

[45] Fayyad U. M., Piatetsky-Shapiro G., Smyth P.: From Data Mining to Knowledge Discovery: An Overview, In: Fayyad U. M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (Eds.): *Advances in Knowledge Discovery and Data Mining*, 1996, 3–33.

[46] Ferre S., Ridoux O.: A file system based on concept analysis. *Proceedings of the 1st International Conference on Computational Logic*, 2000, 1033–1047.

[47] Fu H., Fu H., Njiwoua P., Mephu Nguifo E.: A comparative study of FCA-based supervised classification algorithms. In: Eklund P. (Ed.): *Proc. ICFCA 2004*, *Lecture Notes in Artificial Intelligence* 2961, 313–320.

[48] Fu H., Mephu Nguifo E.: A Parallel Algorithm to Generate Formal Concepts for Large Data. *Proc. ICFCA 2004*, *Lecture Notes in Computer Science* 2961, 394–401.

[49] Ganter B.: *Two basic algorithms in concept analysis.* Technical Report FB4-Preprint No. 831, TH Darmstadt, 1984.

[50] Ganter B., Stumme G., Wille R. (Eds.): Formal Concept Analysis, Foundations and Applications. *Lecture Notes in Computer Science* 3626, Springer, 2005.

[51] Ganter B., Wille R.: *Formal Concept Analysis. Mathematical Foundations.* Springer, Berlin, 1999.

[52] Godin R., Mili H.: Building and maintaining analysis level class hierarchies using Galois lattices. *Proceedings of the 8th Annual Conference on Object Oriented Programming Systems Languages and Applications*, 1993, 394-410.

[53] Goguen J. A.: The logic of inexact concepts. *Synthese* 18(3/4)(1968-9), 325–373.

[54] Goldberg L. A.: *Efficient Algorithms for Listing Combinatorial Structures.* Cambridge University Press, 1993.

[55] Grätzer G. et al.: *General Lattice Theory.* Birkhäuser Basel, 2 edition, 2003.

[56] Guillas S., Bertet K., Visani M., Ogier J. M., Girard N.: Some Links Between Decision Tree and Dichotomic Lattice, In: Belohlavek R., Kuznetsov S. O. (Eds.): *CLA 2008: Proceedings of the Sixth International Conference on Concept Lattices and Their Applications*, 193–205.

[57] Guyon I., Gunn S., Nikravesh M., Zadeh L. A.: *Feature Extraction: Foundations and Applications*, Springer, 2006.

[58] Hájek P.: *Metamathematics of Fuzzy Logic.* Kluwer, Dordrecht, 1998.

[59] Harman H. H.: *Modern Factor Analysis, 2nd Ed.* The Univ. Chicago Press, 1970.

[60] Heath D., Kasif S., Salzberg S.: Induction of Oblique Decision Trees. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*, 1993, 1002–1007.

[61] Hettich S., Bay S. D.: The UCI KDD Archive. University of California, Irvine, School of Information and Computer Sciences, 1999.

[62] Ikeda M., Yamamoto A.: Classification by Selecting Plausible Formal Concepts in a Concept Lattice. *Proc. FCAIR 2013*, 22–35.

[63] Johnson D. S, Yannakakis M., Papadimitriou C. H.: On generating all maximal independent sets. *Information Processing Letters* **27**(3)(1988), 119–123.

[64] Kengue J. F. D., Valtchev P., Djamégni C. T.: A Parallel Algorithm for Lattice Construction. *Proc. ICFCA 2005*, *Lecture Notes in Computer Science* 3403, 249–264.

[65] Kim K. H.: *Boolean Matrix Theory and Applications.* M. Dekker, 1982.

[66] Kirchberg M., Leonardi E., Tan Y. S., Link S., Ko Ryan K. L., Lee B. S.: Formal Concept Discovery in Semantic Web Data. *Proc. ICFCA 2012*, *Lecture Notes in Computer Science* 7278, 164–179.

[67] Kneale W., Kneale M.: *The Development of Logic.* Clarendon Press, Oxford Univ. Press, 1962.

[68] Kneale W., Kneale M.: *The Development of Logic.* Oxford University Press, USA, 1985.

[69] Koester B.: *FooCA – Web Information Retrieval with Formal Concept Analysis.* Verlag Allgemeine Wissenschaft, 2006.

[70] Kohavi R.: A Study on Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence*, 1995, 1137–1145.

[71] Krajca P., Outrata J., Vychodil V.: Advances in algorithms based on CbO. In: Kryszkiewicz M., Obiedkov S. (Eds.): *CLA 2010: Proceedings of the 7th International Conference on Concept Lattices and Their Applications*, 325–337.

[72] Krajca P., Outrata J., Vychodil V.: Computing formal concepts by attribute sorting. *Fundamenta Informaticae* 115(4)(2012), 395–417.

[73] Krajca P., Outrata J., Vychodil V.: Parallel Recursive Algorithm for FCA. In: Belohlavek R., Kuznetsov S. O. (Eds.): *CLA 2008: Proceedings of the Sixth International Conference on Concept Lattices and Their Applications*, 71–82.

[74] Krajca P., Outrata J., Vychodil V.: Parallel Algorithm for Computing Fixpoints of Galois Connections. *Annals of Mathematics and Artificial Intelligence* 59(2)(2010), 257–272.

[75] Krajca P., Outrata J., Vychodil V.: Using frequent closed itemsets for data dimensionality reduction. In: Cook D., Pei J., Wang W., Zaiane O., Wu X. (Eds.): *Proceedings of the ICDM 2011, The 11th IEEE International Conference on Data Mining, 2011*, 1128–1133.

[76] Krajca P., Vychodil V.: Comparison of data structures for computing formal concepts. *Proc. MDAI*, *Lecture Notes in Computer Science* 5861, 2009, 114–125.

[77] Krajca P., Vychodil V.: Distributed algorithm for computing formal concepts using map-reduce framework. *Proc. IDA 2009*, *Lecture Notes in Computer Science* 5772, 333–344.

[78] Krajči S.: Cluster based efficient generation of fuzzy concepts. *Neural Network World* 5(2003), 521–530.

[79] Kuznetsov S.: A fast algorithm for computing all intersections of objects in a finite semi-lattice (Быстрый алгоритм построения всех пересечений объектов из конечной полурешетки, in Russian). *Automatic Documentation and Mathematical Linguistics*, 27(5)(1993), 11–21.

[80] Kuznetsov S.: Interpretation on graphs and complexity characteristics of a search for specific patterns. *Automatic Documentation and Mathematical Linguistics*, 24(1)(1989), 37–45.

[81] Kuznetsov S.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. *PKDD 1999*, 384–391.

[82] Kuznetsov S. O.: Machine learning and formal concept analysis. In: Eklund P. (Ed.): *Proc. ICFCA 2004*, *Lecture Notes in Artificial Intelligence* 2961, 287–312.

[83] Kuznetsov S. O.: On computing the size of a lattice and related decision problems, *Order*, 18(2001), 313–321.

[84] Kuznetsov S., Obiedkov S.: Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Int.*, 14(2/3)(2002), 189–216.

[85] Langdon W. B., Yoo S., Harman M.: Formal Concept Analysis on Graphics Hardware. *Proc. CLA 2011*, 413–416.

[86] Lindig C.: Concept-based component retrieval. *Working Notes of the IJCAI-95 Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs*, 21–25.

[87] Lindig C.: Fast concept analysis. In: Stumme G. (Ed.): *Working with Conceptual Structures—Contributions to ICCS 2000*, 152–161.

[88] Ling Ch. X., Yang Q., Wang J., Zhang S.: Decision Trees with Minimal Costs. *Proc. ICML 2004*, 69–76.

[89] Liu H., Motoda H.: *Computational Methods of Feature Selection.* Chapman and Hall/CRC, 2007.

[90] Liu H., Motoda H.: *Feature Extraction, Construction and Selection: A Data Mining Perspective.* Springer, 1998.

[91] Liu H., Wang X., He J., Han J., Xin D., Shao Z.: Top-down mining of frequent closed patterns from very high dimensional data. *Information Sciences* 179(7)(2009), 899–924.

[92] Mephu Nguifo E., Njiwoua P.: IGLUE: A lattice-based constructive induction system. *Intell. Data Anal.* 5(1)(2001), 73–91.

[93] van der Merwe D., Obiedkov S. A., Kourie D. G.: AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. *Proc. ICFCA 2004*, *Lecture Notes in Artificial Intelligence* 2961, 205–206.

[94] Michalski R. S.: A theory and methodology of inductive learning. *Artificial Intelligence* 20(1983), 111–116.

[95] Miettinen P., Mielikäinen T., Gionis A., Das G., Mannila H.: The discrete basis problem. *PKDD 2006*, 335–346.

[96] Mirkin B.: *Mathematical Classification and Clustering.* Kluwer Academic Publishers, 1996.

[97] Missaoui R., Kwuida L.: What Can Formal Concept Analysis Do for Data Warehouses? *Proc. ICFCA 2009*, *Lecture Notes in Artificial Intelligence* 5548, 58–65.

[98] Mitchell T. M.: *Machine Learning.* McGraw-Hill, 1997.

[99] Murphy P. M., Pazzani M. J.: ID2-of-3: constructive induction of M-of-N concepts for discriminators in decision trees. *Proc. of the Eight Int. Workshop on Machine Learning, 1991*, 183–187.

[100] Murthy S. K.: Automatic construction of decision trees from data. *Data Mining and Knowledge Discovery* 2(1998), 345–389.

[101] Murthy S. K., Kasif S., Salzberg S.: A system for induction of oblique decision trees. *J. of Artificial Intelligence Research* 2(1994), 1–33.

[102] Newman D. J., Hettich S., Blake C. L., Merz C. J.: *UCI Repository of machine learning databases*, `www.ics.uci.edu/~mlearn/MLRepository.html`, University of California, Department of Information and Computer Science, 1998.

[103] Norris E. M.: An algorithm for computing the maximal rectangles of a binary relation. *Journal of ACM* 21(1974), 356–266.

[104] Norris E. M.: An Algorithm for Computing the Maximal Rectangles in a Binary Relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2)(1978), 243–250.

[105] Ore O.: Galois connections. *Trans. Amer. Math. Soc.* 55(1944), 493–513.

[106] Outrata J.: A lattice-free concept lattice update algorithm based on ∗CbO. In: Ojeda-Aciego M., Outrata J. (Eds.): *CLA 2013: Proceedings of the 10th International Conference on Concept Lattices and Their Applications*, 261–274.

[107] Outrata J.: A lattice-free concept lattice update algorithm. *Int. Journal of General Systems* (2015), 21 pp. (to appear).

[108] Outrata J.: Boolean factor analysis for data preprocessing in machine learning. *Proceedings of The Ninth Int. Conf. on Machine Learning and Applications (ICMLA 2010)*, 899–902.

[109] Outrata J.: Inducing decision trees via concept lattices. In: Trappl R. (Ed.): *Cybernetics and Systems 2008: Proceedings of the 19th European Meeting on Cybernetics and Systems Research*, 9–14.

[110] Outrata J.: Preprocessing input data for machine learning by FCA. *CLA 2010: Proceedings of the 7th International Conference on Concept Lattices and Their Applications*, 187–198.

[111] Outrata J., Vychodil V.: Fast Algorithm for Computing Fixpoints of Galois Connections Induced by Object-Attribute Relational Data. *Information Sciences* 185(1)(2012), 114–127.

[112] Pagallo G., Haussler D.: Boolean feature discovery in empirical learning. *Machine Learning* 5(1)(1990), 71–100.

[113] Pasquier N., Bastide Y., Taouil R., Lakhal L.: Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1)(1999), 25–46.

[114] Piramuthu S., Sikora R. T.: Iterative feature construction for improving inductive learning algorithms. *Expert Systems with Applications* 36(2, part 2)(2009), 3401–3406.

[115] Pistori H., Neto J. J.: Decision Tree Induction using Adaptive FSA. *CLEI Electron. J.* 6(1)(2003), 14 pp.

[116] Poelmans J., Kuznetsov S. O., Ignatov D. I., Dedene G.: Formal Concept Analysis in knowledge processing: A survey on models and techniques. *Expert Systems with Applications* 40(16)(2013), 6601–6623.

[117] Pollandt S.: *Fuzzy Begriffe.* Springer, Berlin, 1997.

[118] Priss U.: A Formal Concept Analysis Homepage, `www.upriss.org.uk/fca/fca.html`.

[119] Priss U.: Lattice-based information retrieval. *Knowledge Organization* 27(3)(2000), 132–142.

[120] Quinlan J. R.: *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[121] Quinlan J. R.: Learning decision tree classifiers. *ACM Computing Surveys*, 28(1)(1996), 71–72.

[122] Rokach L., Maimon O. Z.: *Data Mining with Decision Trees: Theory and Applications.* World Scientific Publishing Company, 2008.

[123] Shan B., Qi J., Liu W.: A CUDA-Based Algorithm for Constructing Concept Lattices. *Proc. RSCTC 2012*, *Lecture Notes in Computer Science* 7413, 297–302.

[124] Snelting G.: Reengineering of configurations based on mathematical concept analysis. *ACM Trans. Software Eng. Method.* 5(2)(1996), 146–189.

[125] Snelting G., Tip F.: Reengineering class hierarchies using concept analysis. *ACM Transactions on Programming Languages and Systems* 22(3)(2000), 540–582.

[126] Strok F, Neznanov A.: Comparing and analyzing the computational complexity of FCA algorithms. *SAICSIT '10, Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists.*, 417–420.

[127] Stumme G., Wille R., Wille U.: Conceptual knowledge discovery in databases using formal concept analysis methods. In: Zytkow J. M., Quafofou M. (Eds.): *Principles of Data Mining and Knowledge Discovery*, *Lecture Notes in Artificial Intelligence* 1510, 1998, 450–458.

[128] Surhone L. M., Tennoe M. T., Henssonow S. F.: *Constructive Induction.* Betascript Publishing, 2010.

[129] Surhone L. M., Tennoe M. T., Henssonow S. F.: *Decision-Tree Pruning* Betascript Publishing, 2010.

[130] Tan P. N., Steinbach M., Kumar V.: *Introduction to Data Mining.* Addison Wesley, Boston, MA, 2006.

[131] Tonella P.: Using a concept lattice of decomposition slices for program understanding and impact analysis. *IEEE Transactions on Software Engineering* 29(6)(2003), 495–509.

[132] Utgoff P. E., Brodley C. E.: *Linear machine decision trees.* COINS Technical Report 91-10, Univ. of Massachusetts, MA, 1991.

[133] Valtchev P., Missaoui R., Godin R.: Formal concept analysis for knowledge discovery and data mining: The new challenges. In: *Proc. ICFCA 2004*, *Lecture Notes in Artificial Intelligence* 2961, 352–371.

[134] Valtchev P., Missaoui R., Godin R., Meridji M.: Generating frequent itemsets incrementally: two novel approaches based on Galois lattice theory. *J. Exp. Theor. Artif. Intelligence* 14(2/3)(2002), 115–142.

[135] Vychodil V.: A new algorithm for computing formal concepts. In: Trappl R. (Ed.): *Cybernetics and Systems 2008: Proc. 19th EMCSR*, 15–21.

[136] Wille R.: Restructuring lattice theory: an approach based on hierarchies of concepts. *Ordered Sets* 83(1982), 445–470.

[137] Witten I. H., Frank E.: *Data Mining: Practical machine learning tools and techniques, 2nd Edition.* Morgan Kaufmann, San Francisco, 2005.

[138] Xu B., de Fréin R., Robson E., Foghl M. Ó.: Distributed Formal Concept Analysis Algorithms Based on an Iterative MapReduce Framework. *Proc. ICFCA 2012*, *Lecture Notes in Computer Science* 7278, 292–308.

[139] Yahia S., Jaoua A.: Discovering knowledge from fuzzy concept lattice. In: Kandel A., Last M., Bunke H.: *Data Mining and Computational Intelligence*, 2001, 167–190.

[140] Zacpal J., Sigmund E., Mitáš J., Sklenář V.: Application of the Formal Concept Analysis in Evaluation of Results of ANEWS Questionnaire and Physical Activity of the Czech Regional Centers. *Proc. CLA 2008*, 97–108.

[141] Zaki M. J.: Mining non-redundant association rules. *Data Mining and Knowledge Discovery* 9(2004), 223–248.

Index

Jan Outrata, * December 14, 1978, Šternberk, Czech Republic
email: jan.outrata@upol.cz
web: outrata.inf.upol.cz

Jan Outrata is an assistant professor at the Department of Computer Science, Faculty of Science, Palacký University Olomouc, Czech Republic. He obtained his MSc in Computer Science in 2003, and PhD in Mathematics in 2006, both from Palacký University Olomouc. His research interests include formal concept analysis and relational data analysis, classification and fuzzy relational systems. Jan Outrata has authored or co-authored over 30 papers in these areas in conference proceedings and journals including IEEE Trans. Fuzzy Systems, J. Computer and System Sciences, Int. J. General Systems, or Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems. His hobbies include free software and sports.

Parallel Recursive Algorithm for FCA*

Petr Krajca, Jan Outrata and Vilem Vychodil

Data Analysis and Modelling Laboratory, SUNY Binghamton
Vestal Parkway E, Binghamton, NY 13902–6000, USA
petr.krajca@binghamton.edu, vychodil@binghamton.edu

Department of Computer Science, Palacky University, Olomouc
Tomkova 40, CZ-779 00 Olomouc, Czech Republic
jan.outrata@upol.cz

Abstract. This paper presents a parallel algorithm for computing formal concepts. Presented is a sequential version upon which we build the parallel one. We describe the algorithm, its implementation, scalability, and provide an initial experimental evaluation of its efficiency. The algorithm is fast, memory efficient, and can be optimized so that all critical operations are reduced to low-level bit-array operations. One of the key features of the algorithm is that it avoids synchronization which has positive impacts on its speed and implementation.

1 Introduction

In this paper, we focus on extracting formal concepts, i.e. particular rectangular patterns, in binary object-attribute relational data. The input data, we are interested in, takes form of a two-dimensional data table with rows corresponding to objects, columns corresponding to attributes (features), and table entries being 1's and 0's indicating presence/absence of attributes. Tables like these represent a fundamental form of incidence data. Given a data table, we wish to find all formal concepts [9, 18] present in the table.

There are several algorithms for computing formal concepts, see [13] for an overview and comparison. Among the best known algorithms are Ganter's algorithm [8] and Lindig's algorithm [14] and their variants. Almost all algorithms proposed to date are sequential ones. Since parallel computing is recently gaining interests as hardware manufactures are shifting their focus from improving computing power by increasing clock frequencies to developing processors with multiple cores, there is a need to have scalable parallel algorithms for formal concept analysis (FCA) which can fully utilize the power of such milticore systems and deliver results faster than sequential algorithms. In this paper, we propose a parallel version of an algorithm presented in [16, 17] which is closely related to algorithm Close-by-One [12]. Our algorithm is light weight, fast, memory efficient, and can be implemented so that it uses just static linear data structures utilizing only low-level operations present in arithmetic logic units of contemporary

* Supported by grant No. 1ET101370417 of GA AV ČR and by institutional support, research plan MSM 6198959214.

ISBN 978–80–244–2111–7, Palacký University, Olomouc, 2008.

microchips which significantly improves the performance of its implementations. We describe the algorithm and compare its performance with the other algorithms. We also focus on scalability, i.e. the growth of algorithm's performance with respect to the growing number of processors.

Let us note that computing all formal concepts is interesting not only for FCA itself but has a wide range of applications. For instance, it has been shown in [3] that formal concepts can be used to find optimal factorization of Boolean matrices. In fact, formal concepts correspond with optimal solutions to the discrete basis problem discussed by Miettinen et al. [15]. Finding formal concepts in data tables is therefore an important task.

2 Preliminaries from FCA

In this section we recall basic notions of the formal concept analysis. More details can be found in monographs [9] and [5].

Let $X = \{0, 1, \ldots, m\}$ and $Y = \{0, 1, \ldots, n\}$ be our sets of objects and attributes, respectively. A formal context is a triplet $\langle X, Y, I \rangle$ where $I \subseteq X \times Y$, i.e. I is a binary relation between X and Y, $\langle x, y \rangle \in I$ meaning that object x has attribute y. As usual, we consider a couple of concept-forming operators [9] $^{\uparrow}\colon 2^X \to 2^Y$ and $^{\downarrow}\colon 2^Y \to 2^X$ defined, for each $A \subseteq X$ and $B \subseteq Y$, by

$$A^{\uparrow} = \{y \in Y \mid \text{for each } x \in A\colon \langle x, y \rangle \in I\}, \tag{1}$$

$$B^{\downarrow} = \{x \in X \mid \text{for each } y \in B\colon \langle x, y \rangle \in I\}. \tag{2}$$

By definition (1), $A^{\uparrow}$ is the set of all attributes shared by all objects from A and, by (2), $B^{\downarrow}$ is the set of all objects sharing all attributes from B. Operators $^{\uparrow}\colon 2^X \to 2^Y$ and $^{\downarrow}\colon 2^Y \to 2^X$ defined by (1) and (2) form the so-called Galois connection [9]. A formal concept (in $\langle X, Y, I \rangle$) is any couple $\langle A, B \rangle \in 2^X \times 2^Y$ such that $A^{\uparrow} = B$ and $B^{\downarrow} = A$. If $\langle A, B \rangle$ is a formal concept then A and B will be called the extent and intent of that concept, respectively. The subconcept-superconcept hierarchy $\leq$ is defined as $\langle A_1, B_1 \rangle \leq \langle A_2, B_2 \rangle$ iff $A_1 \subseteq A_2$ (or, iff $B_2 \subseteq B_1$, both the ways are equivalent), see [5, 9] for details.

Remark 1. There is a useful view of formal concepts which is often neglected in literature. Namely, formal concepts in $\langle X, Y, I \rangle$ correspond to maximal rectangles in $\langle X, Y, I \rangle$. In a more detail, any $\langle A, B \rangle \in 2^X \times 2^Y$ such that $A \times B \subseteq I$ shall be called a rectangle in I. Rectangle $\langle A, B \rangle$ in I is a maximal one if, for each rectangle $\langle A', B' \rangle$ in I such that $A \times B \subseteq A' \times B'$, we have $A = A'$ and $B = B'$. Now, it is easily seen that $\langle A, B \rangle \in 2^X \times 2^Y$ is a maximal rectangle in I iff $A^{\uparrow} = B$ and $B^{\downarrow} = A$, i.e. maximal rectangles = formal concepts.

3 Computing Closures

Here we describe a procedure common to both the sequential and parallel versions of our algorithm. It generates a new concept from an existing one by enlarging its intent and shrinking its extent (at the same time).

Procedure ComputeClosure($\langle A, B\rangle, y$)

```
1  for i from 0 upto m do
2  |  set C[i] to 0;
3  end
4  for j from 0 upto n do
5  |  set D[j] to 1;
6  end
7  foreach i in A ∩ rows[y] do
8  |  set C[i] to 1;
9  |  for j from 0 upto n do
10 |  |  if table[i, j] = 0 then
11 |  |  |  set D[j] to 0;
12 |  |  end
13 |  end
14 end
15 return ⟨C, D⟩
```

Representation of the Input Data For the sake of efficiency, we represent each $\langle X, Y, I\rangle$ two ways. First, by a two-dimensional array, denoted *table*, which corresponds with I in the usual sense. That is, the array *table* is filled with 1s and 0s so that $table[i, j] = 1$ iff $\langle i, j\rangle \in I$ and $table[i, j] = 0$ iff $\langle i, j\rangle \notin I$.

The second representation of the data is an array of ordered lists of objects. For each attribute $y \in Y$, we let $rows[y]$ be a list of all objects having the attribute y. Thus, $rows[y]$ contains $x \in X$ iff $\langle x, y\rangle \in I$. In addition to that, the numbers of rows contained in $rows[y]$ will be ordered in the ascending order (this is for the sake of efficiency). For instance, $rows[y] = (2, 4, 7)$ means that the only objects from X having y in I are the objects 2, 4, and 7. The two-dimensional array *table* and the array of lists *rows* will be used by the subsequent algorithms.

All the algorithms we are going to describe will use sets of objects and attributes represented by their characteristic arrays. That is, in case of attributes, a subset $B \subseteq Y = \{0, 1, \ldots, n\}$ will be represented by an $(n + 1)$-element linear array b of 1s and 0s such $b[k] = 1$ iff $k \in B$ (and $b[k] = 0$ iff $k \notin B$). By a slight abuse of notation, we will identify B with b and write $B[k] = 1$ to denote $k \in B$.

Description of the Algorithm If $\langle A, B\rangle$ is a formal concept then due to the monotony of $^{\downarrow\uparrow}$, all the formal concepts whose intents are strictly greater than B can be written as $\left\langle (B \cup C)^{\downarrow}, (B \cup C)^{\downarrow\uparrow}\right\rangle$, where $C \subseteq Y$ is a set of attributes such that there is at least one attribute $y \in Y$ such that $y \in C$ and $y \notin B$. In particular, if we consider $C = \{y\} \subseteq Y$ such that $y \notin B$, then

$$\left\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\right\rangle \tag{3}$$

is a formal concept such that $(B \cup \{y\})^{\downarrow} \subset A$ and $B \subset (B \cup \{y\})^{\downarrow\uparrow}$. This is important from the computational point of view because if we want to compute

$(B \cup \{y\})^{\downarrow}$, it suffices to go exactly through all objects in A having attribute y:

$$(B \cup \{y\})^{\downarrow} = \{x \in A \mid \langle x, y \rangle \in I\} = A \cap \{y\}^{\downarrow}. \tag{4}$$

The common attributes of objects from (4) form the intent of (3). We have just outlined the idea behind our algorithm which generates formal concept (3) given formal concept $\langle A, B \rangle$ and attribute $y \in Y$ which does not belong to B. The corresponding procedure will be called ComputeClosure. It accepts a formal concept $\langle A, B \rangle$ and an attribute $y \notin B$ and produces a new formal concept $\langle C, D \rangle$ which equals to (3). We can show that the algorithm is sound, see [16].

Remark 2. We have used two representations of the input data to establish desired efficiency of computing new formal concepts, i.e. the redundancy in representation is a trade-off for efficiency. The two-dimensional array representation is used to determine which attributes are not present in the intent of the newly computed formal concept (see lines 7–14 of ComputeClosure). The second representation is used to skip rows in which y does not appear. Such rows do not contribute to the closure $(B \cup \{y\})^{\downarrow\uparrow}$, i.e. they can be disregarded. Our representation is most efficient for mid-size data sets (hundreds of attributes + thousands of objects) stored in RAM.

4 Sequential Algorithm

The previous section described how we can efficiently compute a new formal concept (3) given an initial formal concept $\langle A, B \rangle$. In this section we present a simplified version of our sequential algorithm for computing formal concepts [16, 17] which is suitable for parallelization. The main idea behind this algorithm is the same as in case of the algorithm Close-by-One proposed by Kuznetsov in [12].

Listing Formal Concepts in a Unique Order The core of our algorithm is a recursive procedure GenerateFrom which lists all formal concepts using a depth-first search through the space of all formal concepts. The procedure starts with an initial formal concept $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$. During the search, the procedure first generates a new formal concept R by adding attributes to the intent of the current formal concept, i.e. it applies the procedure described in ComputeClosure. Then, it is checked whether R has already been found. If not, it processes R (e.g., prints it on the screen), and proceeds with generating further formal concepts resulting from R by adding attributes to its intent, i.e. here GenerateFrom recursively calls itself with R being the current formal concept.

The key issue here is to have a quick procedure testing whether a newly generated formal concept has been generated before. We generate the formal concepts in a unique order which ensures that each formal concept is processed exactly once. The principle is the following. Let $\langle A, B \rangle$ be a formal concept, $y \in Y$ such that $y \notin B$. Put $D = (B \cup \{y\})^{\downarrow\uparrow}$, i.e. the new formal concept is $\langle (B \cup \{y\})^{\downarrow}, D \rangle$, see (3). Once D is computed using ComputeClosure, we check whether

$$D \cap \{0, 1, \ldots, y-1\} = B \cap \{0, 1, \ldots, y-1\} \tag{5}$$

Procedure GenerateFrom($\langle A, B\rangle, y$)

```
1  process B (e.g., print B on screen);
2  if B = Y or y > n then
3  |  return
4  end
5  for j from y upto n do
6  |  if B[j] = 0 then
7  |  |  set ⟨C, D⟩ to ComputeClosure(⟨A, B⟩, j);
8  |  |  set skip to false;
9  |  |  for k from 0 upto j − 1 do
10 |  |  |  if D[k] ≠ B[k] then
11 |  |  |  |  set skip to true;
12 |  |  |  |  break for loop;
13 |  |  |  end
14 |  |  end
15 |  |  if skip = false then
16 |  |  |  GenerateFrom(⟨C, D⟩, j + 1);
17 |  |  end
18 |  end
19 end
20 return
```

is true. Note that the "$\supseteq$"-part of (5) is trivial. Moreover, (5) is true iff D agrees with B on the attributes $0, 1, \ldots, y-1$. In other words, (5) is true iff, for each $i \in \{0, 1, \ldots, y-1\}$: $i \in D$ iff $i \in B$. Thus, condition (5) expresses the fact that the closure D of $B \cup \{y\}$ does not contain any new attributes which are "before y". Condition (5) will be used to check whether we should process D. If (5) will be false, we will not process D because due to the depth-first search method, D has already been processed.

Description of the Algorithm The algorithm is represented by a procedure GenerateFrom that accepts two arguments. First, a formal concept $\langle A, B\rangle$ represented by characteristic vectors of objects A and attributes B covered by the concept. Second, an attribute y which is the first attribute to be added to B. $\langle A, B\rangle$ serves as an initial concept from which we start generating other formal concepts. After its invocation, GenerateFrom proceeds as follows:

- It processes the formal concept $\langle A, B\rangle$ (e.g., it prints A and B on screen).
- Then, the procedure checks whether B contains all the attributes from Y, i.e. whether B represents the greatest intent, in which case we exit current branch of recursion (lines 2–4).
- The main loop (lines 5–20) iterates over all remaining attributes, starting with the attribute y. In the body of the main loop (lines 6–18), j denotes the current attribute which we are about to add to B. The if-condition at line 6 checks whether j is already present in B. If so, we proceed with another attribute. If j is not present in B, we try to generate new intent from $B \cup \{j\}$ (lines 7–17).

– At line 7, we compute a new formal concept denoted $\langle C, D \rangle$. The loop between lines 9–14 checks whether B and D satisfy condition (5) for y being j. A flag *skip* is initially set to false (line 8). The flag is reset to true iff there is $k < j$ such that B and D disagree on k.
– If *skip* is false, i.e. if D and B agree on all attributes up to $j - 1$, we make a recursive call of the procedure GenerateFrom to compute descendant intents of D, starting with the next attribute $j + 1$ (line 16).

In order to compute all the formal concepts, we invoke GenerateFrom with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$ and $y = 0$ as its arguments. Then, after finitely many steps, the algorithm produces all formal concepts, each of them exactly once. The soundness of the algorithm is proved in [16], cf. also [12].

Relationship to Other Sequential Algorithms Conceptually, GenerateFrom is the same algorithm as Close-by-One proposed by Kuznetsov [12] although there are some technical differences. GenerateFrom can be seen as simpler version of Close-by-One since we are not interested in the order of generated concepts. On the other hand, we utilize ComputeClosure which results to a much better performance. The algorithm is similar to Lindig's algorithm [13, 14] in that it performs a depth-first search through the search space of all formal concepts. The key difference between our algorithm and that proposed by Lindig [14] and its variants is the way how we test that new formal concept has already been found. Lindig's algorithm and its variants use additional data structures to store intents of found formal concepts. Thus, after a new formal concept is computed, Lindig's algorithm looks up for the concept in a data structure, typically a search tree or a hashing table. Our algorithm uses similar idea as Ganter's algorithm [8] to ensure that no concept is generated multiple times, see (5). Compared to Ganter's algorithm, the number of concepts which are computed multiple times and "dropped" is much lower, see [16].

5 Parallel Algorithm

The sequential version of our algorithm, described in previous section, lists all formal concepts using a depth-first search through the space of all formal concepts. Consider a calling tree of the recursive procedure GenerateFrom. The parallel version consists in modification of GenerateFrom so that subtrees of the calling tree are executed simultaneously by independent processes. The problem to solve is, given a process, which subtree(s) will be executed in the process, or, put in other words, how to distribute computed formal concepts among the processes.

Computing Formal Concepts in More Processes In the following we describe our approach for computing formal concepts in a given fixed number P of separate processes running in parallel. In the approach, processes are executing subtrees (of the calling tree of GenerateFrom) containing, in the root node, a call of GenerateFrom for a formal concept generated by a predefined number of

attributes. The number of attributes, denoted by L, is a second parameter of the parallel algorithm. The parameter has an impact on the distribution of computed formal concepts among the processes, see Remark 3 on page 9.

The algorithm, consisting in modification of GenerateFrom, first simulates original sequential GenerateFrom until it reaches the recursion level at which formal concepts generated by $0 < L \leq n$ attributes are to be processed. The initial recursion halts at level which equals L, counting recursion levels from 0 upwards. The formal concepts generated by L attributes, i.e. formal concepts $\langle C, D\rangle = \langle \{y_0, \ldots, y_{L-1}\}^{\downarrow}, \{y_0, \ldots, y_{L-1}\}^{\downarrow\uparrow}\rangle$ such that $y_i \in Y$, are stored in a queue instead of being processed. For each of the P processes there is exactly one queue and the selection of the queue to which we store $\langle C, D\rangle$ is the key point of the algorithm. In fact, by selecting a queue we select a process which will list all formal concepts descendant to $\langle C, D\rangle$. The optimal selection method should distribute all formal concepts to processes equally. This is, however, very hard to achieve since we do not know the distribution of formal concepts in the search space of all formal concepts until we actually compute them all. In the present version of the algorithm we select process r, where r is the total number of stored formal concepts so far modulo the number P of processes.

After filling up the queues, the modified procedure then forks itself into P processes (or, alternatively, runs the following in $P - 1$ new processes too), and in each process the original sequential GenerateFrom is called for each formal concept in the queue of the respective process. This will list all the remaining descendant formal concepts, in parallel.

Description of the Algorithm The algorithm is represented by a procedure ParallelGenerateFrom, the modification of GenerateFrom which accepts one additional argument: the recursion level counter l, which is used to recognize the recursion level L at which formal concepts generated by L attributes are to be stored in a queue rather than processed. After its invocation, ParallelGenerateFrom proceeds as follows:

- Until it reaches the recursion level $L > 0$, the procedure simulates original GenerateFrom (lines 6–24). The code is identical, with two exceptions: first, instead of exiting at line 8 it skips to the point where original GenerateFrom ends and, second, upon each recursive call of itself it increases the recursion level counter l (line 21). In this step it (sequentially) processes all formal concepts generated by up to $L - 1$ attributes.
- When recursion level counter l is equal to L, i.e. the procedure is about to process formal concept $\langle A, B\rangle$ generated by L attributes, it (instead of processing $\langle A, B\rangle$) stores $\langle A, B\rangle$ and y (the attribute to be added to B) to queue $queue[r]$ of selected process r and exits current branch of recursion (lines 2–4). In this step, all formal concepts generated by L attributes are stored in the queues.
- Notice that when ParallelGenerateFrom exits a branch of recursion at line 4, the execution continues at line 22 because line 21 is the only place where ParallelGenerateFrom is recursively called. Therefore, it continues at line

Procedure ParallelGenerateFrom$(\langle A, B\rangle, y, l)$

```
 1 if l = L then
 2 |  select r from 0 to P − 1 (e.g. r = (∑_{s=0}^{P−1} queue[s]) mod P);
 3 |  store (⟨A, B⟩, y) to queue[r];
 4 |  return
 5 end
 6 process B (e.g., print B on screen);
 7 if B = Y or y > n then
 8 |  goto line 25;
 9 end
10 for j from y upto n do
11 |  if B[j] = 0 then
12 |  |  set ⟨C, D⟩ to ComputeClosure(⟨A, B⟩, j);
13 |  |  set skip to false;
14 |  |  for k from 0 upto j − 1 do
15 |  |  |  if D[k] ≠ B[k] then
16 |  |  |  |  set skip to true;
17 |  |  |  |  break for loop;
18 |  |  |  end
19 |  |  end
20 |  |  if skip = false then
21 |  |  |  ParallelGenerateFrom(⟨C, D⟩, j + 1, l + 1);
22 |  |  end
23 |  end
24 end
25 if l = 0 then
26 |  for r from 1 upto P − 1 do
27 |  |  new process
28 |  |  |  while set (⟨C, D⟩, j) to load from queue[r] do
29 |  |  |  |  GenerateFrom(⟨C, D⟩, j);
30 |  |  |  end
31 |  |  end
32 |  end
33 |  while set (⟨C, D⟩, j) to load from queue[0] do
34 |  |  GenerateFrom(⟨C, D⟩, j);
35 |  end
36 end
37 return
```

25 after exiting the loop between line 10–24. Here, it either exits the current branch of recursion (if $l \neq 0$) or continues if the top recursion level ($l = 0$) has been reached (i.e., no more branches of recursion are on the call stack).

– On the top recursion level ($l = 0$), it runs new $P - 1$ processes running in parallel (lines 26, 27) and the last step is performed by the new processes too.
– Finally, still on the top recursion level only, in each process, it calls original GenerateFrom for each formal concept $\langle C, D\rangle$ and attribute j in the queue

of the respective process (lines 28–30 and 33–35). That means, all formal concepts generated by L or more attributes are processed in separate processes running in parallel.

In order to compute all the formal concepts, we invoke ParallelGenerateFrom with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$, $y = 0$ and $l = 0$ as its arguments. Then, after finitely many steps, the algorithm produces all formal concepts, each of them exactly once. The soundness of the algorithm follows directly from the soundness of the sequential version [12, 16] and the fact that processes compute predefined disjoint sub-collections of all formal concepts. This also means that the processes do not interfere with each other and hence the algorithm needs no synchronization. We postpone the proof to the full version of the paper. The parallelization also does not increase the overall theoretical complexity of the algorithm which is the same as for the sequential version.

Remark 3. Note that the parameter L, in addition to the process selection method, also determines the number of formal concepts computed by each process. If $L = 1$, most of the formal concepts (formal concepts descendant to a formal concept generated by a single attribute) are computed by one or two processes. With increasing L, formal concepts are distributed to processes more equally. On the other hand, however, with increasing L more formal concepts are computed sequentially and less in parallel. From our experimentation it seems a good trade-off value is already $L = 2$, where almost all formal concepts (for $n \gg L$) are computed in parallel and are distributed to processes nearly optimally. This will be further discussed in Section 6.

Remark 4. There have been several approaches to parallel algorithms in FCA. For instance, [7] proposes a parallelization of Ganter's algorithm by decomposing the set of all concepts into non-overlapping subsets which are computed simultaneously. Another parallelization of Ganter's algorithm is presented in [2]. The basic idea in [2] is that the lexicographically ordered power set 2^Y is split into p intervals of the same length (p indicates a number of processes). Then, each of the p intervals is executed by an independent process using a serial version of Ganter's algorithm. A different approach is shown, e.g., in [11] where the algorithm is based on dividing the input data into disjoin fragments which are then computed by independent processes. A detailed comparison of the algorithms in terms of their efficiency and scalability is beyond the scope of this paper and will be a subject of future investigation.

6 Experimental Evaluation

We have run several experiments to compare the algorithm with other algorithms for computing formal concepts. In the experiments, we have used Ganter's [8], Lindig's [14] and Berry's [4] algorithms and were interested in the performance of the algorithms measured by the running time. Furthermore, we have run several experiments to compare algorithm performances in dependence on number of

dataset	mushroom	tic-tac-toe	Debian tags	anonymous web
size	8124×119	958×29	14315×475	32710×295
density	19 %	34 %	< 1 %	1 %
our (1 CPU)	6.543	0.092	12.746	65.221
our (2 CPUs)	3.541	0.047	7.710	33.364
our (4 CPUs)	2.343	0.035	4.545	18.520
our (8 CPUs)	1.393	0.029	3.043	11.466
Ganter's	834.409	2.158	1720.827	10039.733
Lindig's	5271.988	14.530	2639.670	13422.643
Berry's	934.507	5.783	1531.944	3615.078

Fig. 1. Performance for selected datasets (seconds)

used CPUs. For the sake of comparison, we have implemented all the algorithms in ANSI C. The experiments were done on otherwise idle 64-bit x86_64 hardware with 8 independent processors (dual processor workstation with Quad-core Intel Xeon Processor E5345, 2.33 GHz, 12 GB RAM).

Note that even the serial version of our algorithm significantly outperforms the most commonly used algorithms for FCA. A detailed comparison can be found in [16]. In this section, we focus primarily on the scalability of our algorithm, i.e., we focus on the speed improvement with growing number of hardware processors.

Our first experiment compares our algorithm with various FCA algorithms using several data tables from the UCI Machine Learning Repository [1], UCI Knowledge Discovery in Databases Archive [10], and our dataset describing packages in the Debian GNU/Linux [6]. The results, along with the information on size and density (percentage of 1s) of used data sets, are depicted in Figure 1. First four rows contain computation times measured in seconds in case of our algorithm which has been run on 1 (sequential version), 2, 4, and 8 hardware processors. From all the graphs and tables we can see that our algorithm (significantly) outperforms all the other algorithms.

We now focus on the scalability of the algorithm, i.e., ability to decrease running time using multiple CPUs (or more precisely CPU cores). We have used selected data sets and various randomly generated data tables. Fig. 2 (left) contains results for selected datasets while Fig. 2 (right) contains results for randomly generated tables with 10000 objects and 5 % density of 1's. By a *relative speedup* which is shown on y-axes in the graphs, we mean the theoretical speedup given by number of hardware processors (e.g., if we have 4 processors, the execution can be 4 times faster). Therefore, the relative speedup is a ratio of running time using a single CPU (the sequential version of the algorithm) and running time using multiple CPU cores. Note that the theoretical maximum of speedup is equal to the number of used CPUs but real speedup is always smaller due to certain overhead caused by managing of multiple threads of computation. Nevertheless, from the point of view of the speedup, we can see from the experiments

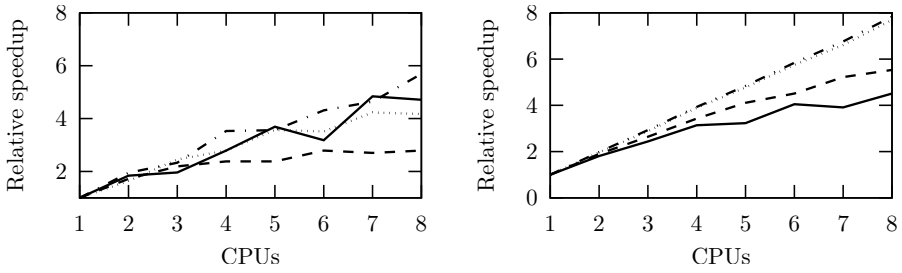

Fig. 2. Relative speedup dependent on various data tables (solid line—mushrooms, dashed line—tic-tac-toe, dotted line—Debian tags, dot-and-dashed line—annonymous web) and used CPU cores (on the left); relative speedup dependent on number of attributes (solid line—50 attributes, dashed line—100 attributes, dotted line—150 attributes, dot-and-dashed line—200 attributes) and used CPU cores measured using randomly generated contexts with 10000 objects and 5 % density (on the right).

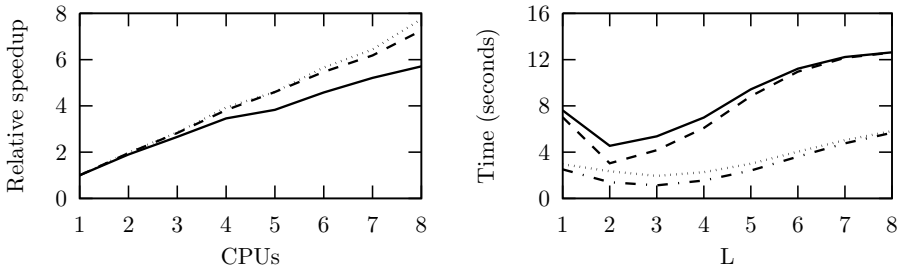

Fig. 3. Relative speedup dependent on density of 1's (solid line—5 %, dashed line—10 %, dotted line—20 %) and used CPU cores (on the left); running time dependent on the argument L (the solid line is for the Debian tags data table and 4 CPUs used, the dashed line is for the Debian tags data table and 8 CPUs used, the dotted lines is for the mushrooms data table and 4 CPUs used and dot-and-dashed lines is for the mushrooms data table and 8 CPUs used) (on the right).

that with growing number of attributes, the real speedup of the algorithm is near its theoretical limits.

In next experiment, that is depicted in Fig. 3 (left), we were focusing on the impact of density of 1's. That is, we have generated data tables with various densities and observed the impact on the scalability. We have used data tables of size 100×10000. Finally, Fig. 3 (right) illustrates the influence of parameter L on various data tables and amounts of CPU cores. The experiments indicate that good choice is $L \in \{2, 3\}$, see Remark 3.

7 Conclusions

We have introduced a parallel algorithm for computing formal concepts in object-attribute data tables. The parallel algorithm is an extension of the serial algo-

rithm we have proposed in [16]. The algorithm consists of a procedure for computing closures and a recursive procedure for computing formal concepts. The main feature of the recursive procedure is that it simulates the sequential one up to a point where the procedure forks into multiple processes and each process computes a disjoint set of formal concepts. Due to our design of the algorithm, there is no need for synchronization which significantly improves efficiency of the algorithm. We have shown that the algorithm is scalable. With growing numbers of CPUs, the speedup of the computation given by increasing number of CPUs is near its theoretical limit. The future research will focus on further refinements of the algorithm and comparison with other approaches.

References

1. Asuncion A., Newman D. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, 2007.
2. Baklouti F., Levy G.: A distributed version of the Ganter algorithm for general Galois Lattices. In: Belohlavek R., Snasel V. (Eds.): *Proc. CLA 2005*, pp. 207–221.
3. Belohlavek R., Vychodil V. On boolean factor analysis with formal concept as factors. *Proceedings of SCIS & ISIS 2006,* pp. 1054–1059, 2006. Tokyo, Japan: Tokyo Institute of Technology.
4. Berry A., Bordat J.-P., Sigayret A. A local approach to concept generation. *Annals of Mathematics and Artificial Intelligence*, **49**(2007), 117–136.
5. Carpineto C., Romano G. *Concept data analysis. Theory and applications.* J. Wiley, 2004.
6. *DAMOL Dataset Repository* (in preparation).
7. Fu H., Mephu Nguifo E.: A Parallel Algorithm to Generate Formal Concepts for Large Data. *ICFCA 2004, LNCS* **2961**, pp. 394–401.
8. Ganter B. *Two basic algorithms in concept analysis.* (Technical Report FB4-Preprint No. 831). TH Darmstadt, 1984.
9. Ganter B., Wille R. *Formal concept analysis. Mathematical foundations.* Berlin: Springer, 1999.
10. Hettich S., Bay S. D.: The UCI KDD Archive University of California, Irvine, School of Information and Computer Sciences, 1999.
11. Kengue J. F. D., Valtchev P., Djamégni C. T.: A Parallel Algorithm for Lattice Construction. *ICFCA 2005, LNCS* **3403**, pp. 249–264.
12. Kuznetsov S.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. *PKDD 1999*, pp. 384–391.
13. Kuznetsov S., Obiedkov S. Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Int.*, **14**(2002), 189–216.
14. Lindig C. Fast concept analysis. *Working with Conceptual Structures– –Contributions to ICCS 2000,* pp. 152–161, 2000. Aachen: Shaker Verlag.
15. Miettinen P., Mielikäinen T., Gionis A., Das G., Mannila H. The discrete basis problem. *PKDD*, pp. 335–346, 2006. Springer.
16. Outrata J., Vychodil V. Fast algorithm for computing maximal rectangles from object-attribute relational data (submitted).
17. Vychodil V.: A new algorithm for computing formal concepts. In: Trappl R. (Ed.): *Cybernetics and Systems 2008: Proc. 19th EMCSR*, 2008, pp. 15–21.
18. Wille R. Restructuring lattice theory: an approach based on hierarchies of concepts. *Ordered Sets*, pp. 445–470, 1982. Dordrecht-Boston.

Ann Math Artif Intell (2010) 59:257–272
DOI 10.1007/s10472-010-9199-5

Parallel algorithm for computing fixpoints of Galois connections

Petr Krajca · Jan Outrata · Vilem Vychodil

Published online: 10 July 2010

Abstract This paper presents a parallel algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. The algorithm results as a parallelization of CbO (Kuznetsov 1999) in which we process disjoint sets of fixpoints simultaneously. One of the distinctive features of the algorithm compared to other parallel algorithms is that it avoids synchronization which has positive impacts on its speed and implementation. We describe the parallel algorithm, prove its correctness, and analyze its asymptotic complexity. Furthermore, we focus on implementation issues, scalability of the algorithm, and provide an evaluation of its efficiency on various data sets.

Keywords Galois connection · Fixpoint · Formal concept · Parallel algorithm

Mathematics Subject Classifications (2010) 03G10 · 62H30 · 11Y16

Supported by research plan MSM 6198959214. Partly supported by grant P103/10/1056 of the Czech Science Foundation.

P. Krajca · J. Outrata · V. Vychodil ( )
Department of Computer Science, Palacky University, Olomouc, Czech Republic
e-mail: vychodil@binghamton.edu

P. Krajca
e-mail: petr.krajca@binghamton.edu

J. Outrata
e-mail: jan.outrata@upol.cz

1 Introduction

We propose a parallel algorithm for computing all fixpoints of Galois connections induced by object-attribute incidence data. The fixpoints, called *formal concepts* [8, 19], represent fundamental rectangular patterns that can be found in the data. Besides their geometrical meaning, the fixpoints can be interpreted as formalizations of natural concepts found in the input incidence data: each formal concept is given by its *extent*, i.e. a set of all objects that fall under the concept, and *intent*, i.e. a set of all attributes (features) that are covered by the concept. The set of all formal concepts equipped with a subconcept–superconcept ordering forms a complete lattice which is commonly called a *concept lattice.* Concept lattices and related incidence structures are thoroughly studied by formal concept analysis—a discipline founded by Rudolf Wille in the early 1980s. Since then, many theoretical results and applications of formal concept analysis (FCA) appeared, see monograph [8] and a recent book [5] for an overview.

The basic task which appears in virtually any application of FCA is to take the input incidence data and compute the set of all formal concepts. The incidence data is represented by a binary relation $I \subseteq X \times Y$ between a set X of objects and a set Y of attributes (features). The data can be depicted by a two-dimensional table with rows corresponding to objects, columns corresponding to attributes, and table entries being ones and zeros indicating presence/absence of attributes. The limiting factor of listing all formal concepts is that the problem is apparently hard as the associated counting problem is $\#P$-complete [13]. Fortunately, if $|I|$ is considerably small, one can get sets of all formal concepts in reasonable time even if X and Y are large. The latter observation resulted in efforts of developing algorithms for FCA specialized on sparse incidence data.

This paper contributes to the family of algorithms for FCA by showing a clear and efficient way to parallelize the computation of concepts by splitting the set of all formal concepts into disjoint subsets which can be computed simultaneously with a minimal overhead. Our motivation for focusing on a parallel algorithm is twofold. First, one of the main problems of FCA is how to deal with large-scale data. The problem has become important recently as FCA is increasingly popular in the data-mining community as a preprocessing technique. Efficient parallelization and distribution over network may help overcome problems with delivering results in a reasonable time (for input data of reasonable size). Second, parallel computing is recently gaining interest as hardware manufactures are shifting their focus from improving computing power by increasing clock frequencies to developing processors with multiple cores. As the multiprocessor systems are becoming more affordable, there will be an increasing pressure to deliver parallel algorithms to better utilize the hardware. From these two points of view, research on parallel algorithms for FCA seems to be promising.

There are several algorithms for computing formal concepts which are closely related to our algorithm. Our algorithm can be seen as a parallelization of a simplified version of CbO [14, 15] and the algorithm proposed by Norris [18]. Our algorithm uses the same canonicity test for avoiding to process the same concept multiple times. This idea also appears in Ganter's algorithm [7] but our algorithm produces

formal concepts in a different order. A detailed comparison will be presented in Section 3.

The paper is organized as follows. In Section 2 we recall notions from formal concept analysis. Section 3 describes the algorithm, shows its correctness, and presents comments on the relationship to other algorithms. Furthermore, in Section 4 we discuss complexity and efficiency issues of the algorithm both theoretically and experimentally. We focus on the scalability of the algorithm, i.e. the growth of its performance with respect to the growing number of processors.

2 Preliminaries and notation

In this section we recall basic notions of the formal concept analysis. More details can be found in monographs [8, 9] and [5]. Let $X = \{0, 1, \ldots, m\}$ and $Y = \{0, 1, \ldots, n\}$ denote finite nonempty sets of objects and attributes, respectively. A formal context is a triplet $\langle X, Y, I\rangle$ where $I \subseteq X \times Y$, i.e. I is a binary relation between X and Y. As usual, given $\langle X, Y, I\rangle$, we consider a pair of concept-forming operators [8] ${}^{\uparrow_I}\colon 2^X \to 2^Y$ and ${}^{\downarrow_I}\colon 2^Y \to 2^X$ defined, for each $A \subseteq X$ and $B \subseteq Y$, by $A^{\uparrow_I} = \{y \in Y \mid \text{for each } x \in A\colon \langle x, y\rangle \in I\}$ and $B^{\downarrow_I} = \{x \in X \mid \text{for each } y \in B\colon \langle x, y\rangle \in I\}$, respectively. If there is no danger of confusion, we omit I and write just ${}^{\uparrow}$ and ${}^{\downarrow}$ instead of ${}^{\uparrow_I}$ and ${}^{\downarrow_I}$, respectively. By a formal concept (in $\langle X, Y, I\rangle$) with extent A and intent B we mean any pair $\langle A, B\rangle \in 2^X \times 2^Y$ such that $A^{\uparrow_I} = B$ and $B^{\downarrow_I} = A$. Thus, formal concepts are fixpoints of the concept-forming operators. The set of all fixpoints of $\langle {}^{\uparrow_I}, {}^{\downarrow_I}\rangle$ will be denoted by $\mathcal{B}(X, Y, I)$. The set $\mathcal{B}(X, Y, I)$ of all formal concepts in $\langle X, Y, I\rangle$ can be equipped with a partial order $\leq$ modeling the subconcept–superconcept hierarchy:

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \text{ iff } A_1 \subseteq A_2 \text{ (or, equivalently, iff } B_2 \subseteq B_1). \tag{1}$$

If $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ then $\langle A_1, B_1\rangle$ is called a subconcept of $\langle A_2, B_2\rangle$. The set $\mathcal{B}(X, Y, I)$ together with $\leq$ defined by (1) form a complete lattice whose structure is described by the Basic Theorem of FCA [8]. For the purpose of illustration, we are going to use the following

Example 1 Consider a formal context $\langle X, Y, I\rangle$ corresponding to the incidence data table from Fig. 1 (left). The concept-forming operators induced by this context have exactly 15 fixpoints (formal concepts) $C_1, \ldots, C_{15}$:

$C_1 = \langle X, \emptyset\rangle$, $C_6 = \langle\{4\}, \{0, 1, 4, 5, 6, 7\}\rangle$, $C_{11} = \langle\{0, 2\}, \{1, 2, 5\}\rangle$,

$C_2 = \langle\{1, 2, 4\}, \{0, 6\}\rangle$, $C_7 = \langle\{1, 2\}, \{0, 3, 6\}\rangle$, $C_{12} = \langle\{0\}, \{1, 2, 4, 5, 7\}\rangle$,

$C_3 = \langle\{2, 4\}, \{0, 1, 5, 6\}\rangle$, $C_8 = \langle\{1\}, \{0, 3, 6, 7\}\rangle$, $C_{13} = \langle\{0, 3, 4\}, \{1, 4, 5\}\rangle$,

$C_4 = \langle\{2\}, \{0, 1, 2, 3, 5, 6\}\rangle$, $C_9 = \langle\{1, 4\}, \{0, 6, 7\}\rangle$, $C_{14} = \langle\{0, 4\}, \{1, 4, 5, 7\}\rangle$,

$C_5 = \langle\emptyset, Y\rangle$, $C_{10} = \langle\{0, 2, 3, 4\}, \{1, 5\}\rangle$, $C_{15} = \langle\{0, 1, 4\}, \{7\}\rangle$.

I	0	1	2	3	4	5	6	7
0	0	1	1	0	1	1	0	1
1	1	0	0	1	0	0	1	1
2	1	1	1	1	0	1	1	0
3	0	1	0	0	1	1	0	0
4	1	1	0	0	1	1	1	1

I	0	1	2	3	4	5	6	7
0	0	1	1	0	1	1	0	1
1	1	0	0	1	0	0	1	1
2	1	1	1	1	0	1	1	0
3	0	1	0	0	1	1	0	0
4	1	1	0	0	1	1	1	1

Fig. 1 Formal context (*left*) and maximal rectangles (*right*) corresponding to C_9 and C_{13}

Hence, $\mathcal{B}(X, Y, I) = \{C_1, \ldots, C_{15}\}$. If we equip $\mathcal{B}(X, Y, I)$ with the partial order (1), the resulting structure is the concept lattice of $\langle X, Y, I\rangle$.

Note that formal concepts in $\langle X, Y, I\rangle$ correspond to so-called maximal rectangles [8] in $\langle X, Y, I\rangle$, cf. Fig. 1 (right).

3 Algorithm for computing all fixpoints

In this section we describe the algorithm for computing all fixpoints of a Galois connection. We start by describing a subroutine which can be seen as a serial version of the algorithm. The main idea behind the serial subroutine of our algorithm is the same as in case of the algorithm Close-by-One (CbO) proposed by Kuznetsov in [15]. The parallel algorithm can be seen as several instances of the serial version working simultaneously on disjoint subsets of concepts. Since Galois connections induced by formal contexts are in fact the most general ones, we focus on fixpoints of $\langle {}^{\uparrow_I}, {}^{\downarrow_I}\rangle$ for a given formal context $\langle X, Y, I\rangle$ such that $X = \{0, 1, \ldots, m\}$ and $Y = \{0, 1, \ldots, n\}$.

Algorithm 1 Procedure GenerateFrom $(\langle A, B\rangle, y)$

```
   Input: formal concept ⟨A, B⟩ and a number y ∈ Y ∪ {n + 1} such that y ∉ B
 1 process ⟨A, B⟩ (e.g., print ⟨A, B⟩ on the screen);
 2 if B = Y or y > n then
 3 |   return
 4 end
 5 for j from y upto n do
 6 |   if j ∉ B then
 7 |   |   set C to A ∩ {j}↓;
 8 |   |   set D to C↑;
 9 |   |   if B ∩ Y_j = D ∩ Y_j then
10 |   |   |   GenerateFrom(⟨C, D⟩, j + 1);
11 |   |   end
12 |   end
13 end
14 return
```

The core of the serial algorithm is a recursive procedure GenerateFrom, see Algorithm 1, which lists all formal concepts using a depth-first search through the space of all formal concepts. The procedure accepts a formal concept $\langle A, B\rangle$ (an

initial formal concept) and an attribute $y \in Y$ (first attribute to be processed) as its arguments. The procedure recursively descends through the space of formal concepts, beginning with the formal concept $\langle A, B\rangle$.

When invoked with $\langle A, B\rangle$ and $y \in Y$, GenerateFrom first processes $\langle A, B\rangle$ (e.g., prints it on the screen or stores it in a data structure, see line 1 of Algorithm 1) and then it checks its halting condition, see lines 2–4. According to the halting condition, the computation stops either when $\langle A, B\rangle$ equals $\langle Y^{\downarrow}, Y\rangle$ (the least formal concept has been reached) or $y > n$ (there are no more remaining attributes to be processed). Otherwise, the procedure goes through all attributes $j \in Y$ such that $j \geq y$ which are not contained in the intent B (see lines 5 and 6). For each $j \in Y$ having these properties, a new pair $\langle C, D\rangle \in 2^X \times 2^Y$ such that

$$\langle C, D\rangle = \langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow}\rangle \tag{2}$$

is computed (lines 7 and 8). One can show that $\langle C, D\rangle$ is always a formal concept such that $B \subset D$ (see Remark 1 below). After obtaining $\langle C, D\rangle$, the algorithm checks whether it should continue with $\langle C, D\rangle$ by recursively calling GenerateFrom or whether $\langle C, D\rangle$ should be "skipped". The test is based on comparing $B \cap Y_j = D \cap Y_j$ where $Y_j \subseteq Y$ is defined as follows:

$$Y_j = \{y \in Y \mid y < j\}. \tag{3}$$

The role of the test (see lines 9–11) is to prevent processing the same formal concept multiple times. In the sequel we prove that GenerateFrom computes formal concepts in a unique order which ensures that each formal concept is processed exactly once.

Remark 1 If $\langle A, B\rangle$ is a formal concept then $\langle C, D\rangle$ computed in lines 7 and 8 of Algorithm 1 is also a formal concept such that $B \subset D$ and $C \subset A$ provided that $j \notin B$. Indeed, $D = C^{\uparrow}$ by definition. Moreover, $C = A \cap \{j\}^{\downarrow} = B^{\downarrow} \cap \{j\}^{\downarrow} = (B \cup \{j\})^{\downarrow}$. Since $^{\downarrow\uparrow\downarrow}$ equals $^{\downarrow}$, we get $D^{\downarrow} = C^{\uparrow\downarrow} = (B \cup \{j\})^{\downarrow\uparrow\downarrow} = (B \cup \{j\})^{\downarrow} = C$, i.e. $\langle C, D\rangle$ is a formal concept. The facts $B \subset D$ and $C \subset A$ follow from properties of the concept-forming operators $^{\downarrow}$ and $^{\uparrow}$ using $j \notin B$.

In order to prove the correctness of Algorithm 1, we introduce so-called derivations which will correspond to recursive invocations of the procedure GenerateFrom. Later, the derivations will be used to describe the parallel algorithm.

Definition 1 (Derivations of Formal Concepts) Let $\langle X, Y, I\rangle$ be a formal context with $Y = \{0, \ldots, n\}$. For formal concepts $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle \in \mathcal{B}(X, Y, I)$ and integers $y_1, y_2 \in Y \cup \{n+1\}$ let $\langle\langle A_1, B_1\rangle, y_1\rangle \vdash \langle\langle A_2, B_2\rangle, y_2\rangle$ denote that for $m = y_2 - 1$ the following conditions

(i) $m \notin B_1$,
(ii) $y_1 < y_2$,

(iii) $B_2 = (B_1 \cup \{m\})^{\downarrow\uparrow}$, and
(iv) $B_1 \cap Y_m = B_2 \cap Y_m$, where Y_m is defined by (3)

are all satisfied. A *derivation of* $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ *of length* $k+1$ is any sequence

$$\langle\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle, 0\rangle = \langle\langle A_0, B_0\rangle, y_0\rangle, \langle\langle A_1, B_1\rangle, y_1\rangle, \ldots, \langle\langle A_k, B_k\rangle, y_k\rangle = \langle\langle A, B\rangle, y_k\rangle \quad (4)$$

such that $\langle\langle A_i, B_i\rangle, y_i\rangle \vdash \langle\langle A_{i+1}, B_{i+1}\rangle, y_{i+1}\rangle$ for each $i = 0, \ldots, k-1$. If $\langle A, B\rangle$ has a derivation of length k we say that $\langle A, B\rangle$ is *derivable in k steps.*

It is easily seen that $\langle\langle A, B\rangle, y\rangle \vdash \langle\langle C, D\rangle, k\rangle$ iff the invocation of GenerateFrom$(\langle A, B\rangle, y)$ causes GenerateFrom$(\langle C, D\rangle, k)$ to be called in line 10. Indeed (i) ensures that the condition in line 6 of Algorithm 1 is satisfied, (ii) corresponds to the fact that the loop between lines 5–13 goes from y upwards, (iii) is the intent computed in line 8, and (iv) is true iff the condition in line 9 is true. Algorithm 1 and derivations are further demonstrated by the following example.

Example 2 Consider the formal context $\langle X, Y, I\rangle$ from Fig. 1 (left). According to Example 1, denote $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle = \langle X, \emptyset\rangle$ by C_1. If GenerateFrom$(C_1, 0)$ is called, j goes over all attributes from Y, starting with $y = 0$, see line 5. For $j = 0$, new formal concept $\langle C, D\rangle$ with $C = \{1, 2, 4\}$ and $D = \{0, 6\}$ is computed (lines 7 and 8). Denote the concept by C_2. Since $D \cap Y_0 = \emptyset = B \cap Y_0$, i.e. the test in line 9 is successful, GenerateFrom$(C_2, 1)$ is invoked. In terms of derivations, we have $\langle C_1, 0\rangle \vdash \langle C_2, 1\rangle$. During the invocation of GenerateFrom$(C_2, 1)$, j goes over all attributes starting with 1. For $j = 1$, we get $C = \{2, 4\}$, $D = \{0, 1, 5, 6\}$. Since $\{0, 6\} \cap \{0\} = \{0, 1, 5, 6\} \cap \{0\}$, the test is successful and GenerateFrom$(C_3, 2)$ is invoked where C_3 denotes $\langle\{2, 4\}, \{0, 1, 5, 6\}\rangle$. Thus, $\langle C_2, 1\rangle \vdash \langle C_3, 2\rangle$. In a similar way we get $\langle C_3, 2\rangle \vdash \langle C_4, 3\rangle$ and $\langle C_4, 3\rangle \vdash \langle C_5, 5\rangle$. When GenerateFrom$(C_5, 5)$ is invoked, all attributes are already present in the intent, i.e., the invocation of GenerateFrom$(C_5, 5)$ is terminated and the computation goes back to GenerateFrom$(C_4, 3)$ with $j \geq 5$. Since the intent of C_4 contains both 5 and 6, we continue with $j = 7$, for which we obtain a formal concept $\langle C, D\rangle = \langle \emptyset, Y\rangle = C_5$ which has already been found. In this case, the test in line 9 fails because $B \cap Y_7 = \{0, 1, 2, 3, 5, 6\} \neq \{0, 1, 2, 3, 4, 5, 6\} = D \cap Y_7$. Therefore, the invocation of GenerateFrom$(C_4, 3)$ is terminated because $j = n = 7$ is the last attribute and the computation proceeds with GenerateFrom$(C_3, 2)$ with $j \geq 3$. For $j = 3$, we obtain a concept $\langle C, D\rangle = C_4$ which has also been found and the test fails because $B \cap Y_3 = \{0, 1\} \neq \{0, 1, 2\} = D \cap Y_3$. For $j = 4$, we obtain a new concept $\langle C, D\rangle = \langle\{4\}, \{0, 1, 4, 5, 6, 7\}\rangle = C_6$ which has not been considered so far. The test succeeds, GenerateFrom$(C_6, 5)$ is invoked, meaning $\langle C_3, 2\rangle \vdash \langle C_6, 5\rangle$, and the computation continues in a similar way as before.

Remark 2 The computation of Algorithm 1 and the corresponding derivations can be depicted by a tree as in Fig. 2. The tree contains two types of nodes. Nodes represented by pairs $\langle C_i, y_i\rangle$ represent arguments of GenerateFrom, i.e. each node of this type represents an invocation of GenerateFrom. Leaf nodes denoted by black squares represent computed concepts for which the test in line 9 fails. Each edge in the tree is labeled by the current value of j which is used to compute a (new) formal concept, see lines 7 and 8. We call such a tree a call tree of GenerateFrom for given

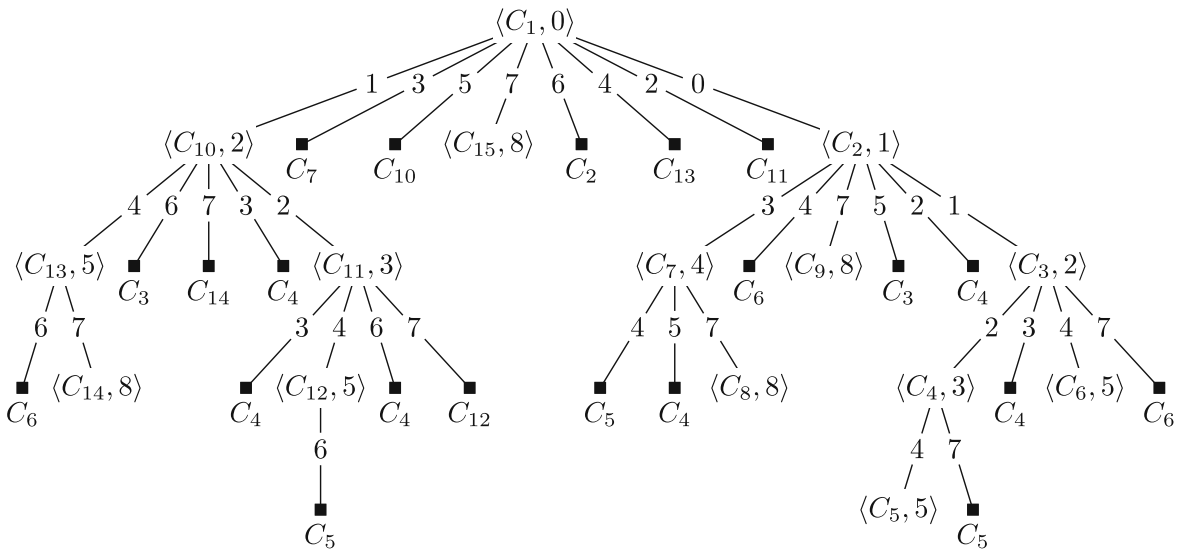

Fig. 2 Example of a call tree for GENERATEFROM($\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle, 0$) with input data from Fig. 1

$\langle X, Y, I \rangle$. A path from the root of the tree to any node labeled by $\langle C_i, y_i \rangle$ corresponds to a derivation of $\langle C_i, y_i \rangle$. Later, we prove that the nodes labeled by $\langle C_i, y_i \rangle$ are always in a one-to-one correspondence with formal concepts in $\mathcal{B}(X, Y, I)$, showing that the algorithm is correct.

The following assertions show the existence and uniqueness of derivations.

Lemma 1 (Existence of Derivations) *For each formal concept* $\langle A, B \rangle \in \mathcal{B}(X, Y, I)$ *there is a derivation* (4) *such that* $y_i = m_i + 1$ *where*

$$m_i = \min\{y \in B \mid y \notin B_{i-1}\} \tag{5}$$

for each $0 < i \leq k$.

Proof We prove by induction over i that $\langle \langle A_0, B_0 \rangle, y_0 \rangle, \ldots, \langle \langle A_i, B_i \rangle, y_i \rangle$ is a derivation. Assume that the claim holds for $0, \ldots, i-1 < k$. We prove that it holds for i. Since $i - 1 < k$, $B \setminus B_{i-1} \neq \emptyset$. Therefore, m_i given by (5) and consequently $y_i = m_i + 1$ are well defined. Put $B_i = (B_{i-1} \cup \{m_i\})^{\downarrow\uparrow}$ and $A_i = A_{i-1} \cap \{m_i\}^{\downarrow}$. We now prove that $\langle \langle A_{i-1}, B_{i-1} \rangle, y_{i-1} \rangle \vdash \langle \langle A_i, B_i \rangle, y_i \rangle$ by checking Definition 1 (i)–(iv). Using (5), $y_i - 1 = m_i \notin B_{i-1}$, i.e. (i) is true. In order to prove (ii), we check that $m_{i-1} < m_i$. By contradiction, assume that $m_i \leq m_{i-1}$. Obviously, $m_{i-1} \neq m_i$ because $m_i \notin B_{i-1}$ and $m_{i-1} \in B_{i-1}$. Thus, assume $m_i < m_{i-1} \neq 0$. Since $m_i \notin B_{i-1}$ and $B_{i-2} \subset B_{i-1}$, we get $m_i \notin B_{i-2}$. Using the induction hypothesis, $m_{i-1} = \min\{y \in B \mid y \notin B_{i-2}\}$ which contradicts the facts that $m_i < m_{i-1}$ and $m_i \notin B_{i-2}$, proving (ii). Condition (iii) agrees with the definition of B_i. It remains to check that $B_{i-1} \cap Y_{m_i} = B_i \cap Y_{m_i}$. Since $B_{i-1} \subset B_i = (B_{i-1} \cup \{m_i\})^{\downarrow\uparrow} \subseteq B$, m_i is a minimum attribute such that $m_i \in B_i$ and $m_i \notin B_{i-1}$. That is, for each $y < m_i$, $y \in B_{i-1}$ iff $y \in B_i$. The latter is equivalent to $B_{i-1} \cap Y_{m_i} = B_i \cap Y_{m_i}$, showing (iv). □

Lemma 2 (Uniqueness of Derivations) *Each formal concept* $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ *has at most one derivation.*

Proof According to Lemma 1, we prove that each derivation of $\langle A, B\rangle$ equals to (4) where $y_i = m_i + 1$ and m_i are given by (5). By contradiction, let

$$\langle\langle A'_0, B'_0\rangle, y'_0\rangle, \langle\langle A'_1, B'_1\rangle, y'_1\rangle, \ldots, \langle\langle A'_l, B'_l\rangle, y'_l\rangle$$

be another derivation of $\langle A, B\rangle$. Let i by the index such that $y_j = y'_j$ for all $j < i$ and $y_i \neq y'_i$. It is easily seen that $A_j = A'_j$ and $B_j = B'_j$ for all $j < i$. Furthermore, for m_i given by (5), we get $m_i \in B_i$. The observations that $m_i \notin B_j = B'_j$ for all $j < i$ and that m_i is the minimum attribute in $B \setminus B_{i-1} = B \setminus B'_{i-1}$ yield $m_i \notin B'_i$ because otherwise $B'_{i-1} \cap Y_{y'_i - 1} = B'_i \cap Y_{y'_i - 1}$ would be violated. On the other hand, $m_i \in B = B'_l$, i.e. there must be an index $j > i$ such that $m_i \in B'_j$ and $m_i \notin B'_h$ for all $h < j$. In addition to that, we have $m_i < y'_i - 1 < y'_j - 1$. Therefore, $m_i \in B'_j \cap Y_{y'_j - 1}$ and $m_i \notin B'_{j-1} \cap Y_{y'_j - 1}$, contradicting the fact that $B'_{j-1} \cap Y_{y'_j - 1} = B'_j \cap Y_{y'_j - 1}$. □

We now get the following consequence of Lemmas 1 and 2:

Theorem 1 (Correctness of Algorithm 1) *When invoked with* $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$ *and* $y = 0$, *Algorithm 1 derives all formal concepts in* $\langle X, Y, I\rangle$, *each of them exactly once.* □

Remark 3 Algorithm 1 can be seen as a simplified version of CbO [14, 15]. We formulate the algorithm by a recursive procedure GenerateFrom rather than by backtracking as it is used in [15]. This has several benefits. First, GenerateFrom is much closer to the actual implementation than the abstract description from [15]. Second, there is no need for explicit labeling of attributes which have been processed, see [15], because each invocation of GenerateFrom has all the necessary information in a local variable j. When computing new closures, we improve the efficiency of the algorithm by going through only a subset of all attributes from Y, see line 5 of Algorithm 1. Finally, there is no need to build the CbO-tree [15] as a data structure. The CbO-tree corresponds to the recursive invocations of GenerateFrom: derivations from Definition 1 correspond to canonical paths in the CbO-tree, see [15]. Paths which are not canonical according to [15] can be seen as paths from the root node of the call tree of GenerateFrom to nodes labeled by black squares, see Fig. 2.

Ganter's algorithm [7] is also closely related to our algorithm but it lists formal concepts in a different order. On the other hand, our algorithm can be easily modified to produce formal concepts in the same order with a slight loss of the performance. Indeed, during each invocation of GenerateFrom$(\langle A, B\rangle, y)$ it suffices to (i) build a list $\mathcal{L}$ of all concepts $\langle A_i, B_i\rangle$ such that $\langle\langle A, B\rangle, y\rangle \vdash \langle\langle A_i, B_i\rangle, j_i\rangle$ $(j_i > y)$ without invoking GenerateFrom$(\langle A_i, B_i\rangle, j_i)$, then (ii) sort the list $\mathcal{L}$ according to the lexicographic order [7] on the intents B_i, and (iii) recursively invoke GenerateFrom$(\langle A_i, B_i\rangle, j_i)$ for all $\langle A_i, B_i\rangle$ in the sorted list $\mathcal{L}$ according to the lexicographic order.

We now turn our attention to the parallel algorithm. Assume that we have P *independent processors* which can execute instructions simultaneously. These may represent separate computers in a network or multiple processors in a system with

shared memory. We assume that each processor has access to the context $\langle X, Y, I\rangle$. Since $\langle X, Y, I\rangle$ is not altered during the computation, each processor can have its own copy of $\langle X, Y, I\rangle$ or share one copy among multiple processors (in systems with shared memory).

Algorithm 2 Procedur ParallelGenerateFrom ($\langle A, B\rangle, y, l$)

```
   Input: formal concept ⟨A, B⟩,
          number y ∈ Y ∪ {n + 1} such that y ∉ B,
          level of recursion L ≥ 2,
          number of processors P ≥ 1, and
          counter l such that 1 ≤ l ≤ L
 1 if l = L then
 2 |   select r ∈ {1, ..., P};
 3 |   store ⟨⟨A, B⟩, y⟩ to queue_r;
 4 |   return
 5 end
 6 process ⟨A, B⟩ (e.g., print ⟨A, B⟩ on screen);
 7 if not (B = Y or y > n) then
 8 |   for j from y upto n do
 9 |   |   if j ∉ B then
10 |   |   |   set C to A ∩ {j}↓;
11 |   |   |   set D to C↑;
12 |   |   |   if B ∩ Y_j = D ∩ Y_j then
13 |   |   |   |   ParallelGenerateFrom(⟨C, D⟩, j + 1, l + 1)
14 |   |   |   end
15 |   |   end
16 |   end
17 end
18 if l = 1 then
19 |   for r from 1 upto P do
20 |   |   with processor r
21 |   |   |   foreach ⟨⟨C, D⟩, j⟩ ∈ queue_r do
22 |   |   |   |   GenerateFrom(⟨C, D⟩, j);
23 |   |   |   end
24 |   |   end
25 |   end
26 |   wait for all processors
27 end
28 return
```

The parallelization we propose consists in modification of GenerateFrom so that particular subtrees of the call tree are computed simultaneously by P processors. The idea is best explained when we consider a call tree like the one in Fig. 2. Recall that GenerateFrom is a recursive procedure and its invocations during the computation agree with the nodes labeled $\langle C_i, y_i\rangle$ in the tree. Moreover, the order in which the concepts are processed can be read directly from the call tree. It suffices to go through the $\langle C_i, y_i\rangle$ nodes in the depth-first order following the labels of edges from smallest to biggest numbers. At any level of the call tree, we obtain a set of nodes which are root nodes of disjoint subtrees. For instance, in Fig. 2, the second level of the call tree contains nodes $\langle C_{10}, 2\rangle$, $\langle C_{15}, 8\rangle$, and $\langle C_2, 1\rangle$. Two of the nodes are root nodes of nontrivial subtrees which may be processed independently by two processors. This suggests to modify GenerateFrom so that is goes through the call tree only up to a certain predefined level L and then it lets P independent processors compute the remaining concepts descendant to those on the Lth level. In terms of derivations, see Definition 1, the algorithm first processes all concepts which are derivable in less

than L steps. The remaining concepts are computed in parallel. Therefore, a parallel procedure for computing concepts can be summarized by three consecutive stages:

Stage 1: Compute and process all concepts that are derivable in less than L steps.

Stage 2: Store all concepts derivable in L steps in P independent queues.

Stage 3: Initiate P processors and run the parallel computation: (i) let each of the processors take exactly one of the queues; (ii) let each processor compute all concepts (using Algorithm 1) beginning with those in its queue.

A parallel algorithm following this idea is represented by procedure ParallelGenerateFrom, see Algorithm 2. It is important to note that Algorithm 2 has two parameters which are constant during the computation: $P \geq 1$ (*number of processors*) and $L \geq 2$ (*level of recursion*, i.e. the maximum length of derivations which are computed sequentially in Stage 1). The choice of values of P and L has an influence of the practical performance of the algorithm. This issue will be addressed later. Procedure ParallelGenerateFrom is a modification of GenerateFrom and accepts one additional argument: a *counter* l which goes from 1 up to L and is used to indicate lengths of derivations that are processed in Stage 1.

After its invocation, ParallelGenerateFrom proceeds as follows: The procedure simulates the original GenerateFrom until is reaches the recursion level L, see the code between lines 1–17. This agrees with Stage 1 as outlined above. There are two technical differences between GenerateFrom and ParallelGenerateFrom:

- ParallelGenerateFrom increases the counter l upon each invocation, see line 13. Obviously, if the procedure is initially called with $l = 1$ then during the computation l is always equal to the current recursion level (call tree level). In addition to that, formal concepts that are processed in line 6 are exactly the concepts derivable in less than L steps.
- Instead of returning from the recursion, see the condition in line 7, the procedure continues to the point where the original GenerateFrom ends. This step is taken because ParallelGenerateFrom has to initiate the parallel computation after the first two stages are finished, see lines 18–27.

When l equals L, ParallelGenerateFrom has reached the level of recursion at which the serial algorithm stops, entering the Stage 2. In other words, $l = L$ means that the current formal concept $\langle A, B \rangle$ is derivable in L steps. Instead of processing $\langle A, B \rangle$ in line 6, the procedure performs the code between lines 2–4, i.e., it selects one of the queues numbered $1, \ldots, P$, stores $\langle \langle A, B \rangle, y \rangle$ in the queue, and exits this branch of recursion. During this stage of computation, all formal concepts derivable in L steps are stored in the queues.

Notice that the limit condition in line 1 also ensures that there are only finitely many recursive invocations of ParallelGenerateFrom. Since $L \geq 2$ and the initial value of the counter l equals 1, the initial invocation of ParallelGenerateFrom is never terminated in line 4. As a consequence, after finitely many steps, the initial invocation of ParallelGenerateFrom gets to the line 18. Here, the condition succeeds because $l = 1$. Thus, the initial invocation proceeds with lines 19–26 which take care of initiating the parallel computation: each processor goes over all $\langle \langle A, B \rangle, y \rangle$ in its queue and invokes the serial procedure GenerateFrom with $\langle A, B \rangle$ and y as its arguments. The only synchronization that is used in the algorithm is that the initial invocation waits until all processors finish the computation, see line 26. Also note that

the condition in line 1 ensures that the parallel computation will be initiated exactly once because there is only one invocation of ParallelGenerateFrom with $l = 1$.

Remark 4 The key issue with Algorithm 2 is how to distribute formal concepts derivable in L steps into P queues. In fact, by selecting a queue in which we put $\langle\langle C, D\rangle, y\rangle$ we select a processor which will list all formal concepts descendant to $\langle C, D\rangle$. The optimal selection method should distribute all formal concepts to processors uniformly. This is, however, very hard to achieve since we do not know the distribution of formal concepts in the search space of all formal concepts until we actually compute them all and reveal the structure of the call tree. In the present version of the algorithm we select $queue_r$ based on a simple round-robin principle: the index r is computed as $r = (N \bmod P) + 1$ where N denotes the number of formal concepts stored so far. This principle, albeit simple, turned out to be efficient for both the real-world datasets and randomly generated data, see Section 4.

Our algorithm can be seen as having two parts: first, a part which distributes concepts into queues and, second, a part which runs several instances of the ordinary Close-by-One in parallel. Because of this reliance on CbO, we call our algorithm *Parallel Close-by-One* (*PCbO*). The following assertion shows correctness of PCbO:

Theorem 2 (Correctness of PCbO) *When invoked with* $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$, $y = 0$, *and* $l = 1$, *Algorithm* 2 *derives all formal concepts in* $\langle X, Y, I\rangle$, *each of them exactly once.*

Proof The correctness is a consequence of properties of derivations, see Lemmas 1 and 2. First, it is easy to observe that Algorithm 2 finishes after finitely many steps. Moreover, each concept that is derivable in less than L steps is processed in the first stage, each of them is processed exactly once. This follows from the fact that ParallelGenerateFrom simulates GenerateFrom. If a concept is derivable in $> L$ steps, it will be computed by one of the independent processors. Indeed, let (4) be the derivation of $\langle A, B\rangle$ where $k + 1 > L$. Then the $(L-1)$th element $\langle\langle A_{L-1}, B_{L-1}\rangle, y_{L-1}\rangle$ of the derivation (4) will be put in one of the queues, say $queue_r$, in the second stage of the algorithm because $\langle A_{L-1}, B_{L-1}\rangle$ is derivable in L steps. Therefore, $\langle A, B\rangle$ will be computed by the processor r. In addition to that, $\langle A, B\rangle$ will be computed exactly once on the account of Lemma 2. □

Remark 5 Let us comment on the role of P and L which influence Algorithm 2. Both the parameters have an impact on the distribution of computed formal concepts among the processors. Note that the practical range of the parameter P is somewhat limited by the hardware on which we run the algorithm (e.g., we are limited by hardware processors or network nodes). On the other hand, L can be set to any value ≥ 2. The performance of the algorithm in dependence of the value of L is experimentally evaluated in Section 4. According to our observations, if $L = 2$, most of the formal concepts are computed by one or two processors. With increasing L, formal concepts are distributed to processors more equally. On the other hand, large values of L tend to degenerate the parallel computation. For instance, if $L \geq |Y| + 1$ then all concepts will be computed in the first (sequential) stage because the depth of the call tree is at most $|Y| + 1$. From our experiments it seems that on average, a good trade-off value is already $L = 3$ provided that $|Y|$ is large. In such a case, almost all

formal concepts are computed in parallel and are distributed among the processors nearly optimally.

Example 3 We illustrate the influence of P and L on how Algorithm 2 computes the concepts. Consider a formal context $\langle X, X, \neq \rangle$ where $|X| = 5$. The corresponding $\mathcal{B}(X, X, \neq)$ is isomorphic to the Boolean algebra 2^5. Figure 3 contains results for four combinations of values of P and L. Each of the diagrams in Fig. 3 depicts the Hasse diagram of the concept lattice where nodes denoted by black circles correspond to concepts processed during the initial sequential stage. Nodes denoted by numbers are processed by independent processors of the corresponding numbers. In case of $P = 2$ and $L = 2$, only the topmost concept is processed in the first stage. During the second stage, three concepts are put in the queue of the first processor, the remaining two concepts are put in the queue of the second processor. The total number of concepts that are processed by the two processors are 21 and 10, respectively. If $P = 2$ and $L = 3$ (second diagram), the concepts are distributed among the processors more equally: 16 and 10. A similar situation applies for $P = 3$ where we have 18, 9, and 4 concepts processed by three processors in case of $L = 2$ and 11, 10, and 5 in case of $L = 3$, see last two diagrams.

Remark 6 The parallel computation of Algorithm 2 can be degenerate, meaning that in certain situations, only one of the P processors is computing all the remaining concepts while other processors are idle. Such a situation occurs iff the Lth level of the call tree contains at most one node $\langle C_i, y_i \rangle$. In particular, the situation occurs when $\mathcal{B}(X, Y, I)$ is isomorphic to an ordinal sum $\mathbf{L}_1 \oplus \mathbf{L}_2$ of a lattice $\mathbf{L}_1$ and an n-element chain $\mathbf{L}_2$ where n equals L (the recursion level), see Fig. 4. Such pathologic situations can be (partially) avoided by modifying the condition in line 1 of Algorithm 2 so that is checks whether at least a given number of queues are nonempty. More details on the utilization of processors can be found in Section 4.

Let us conclude this section with bibliographical remarks on existing approaches to parallel algorithms in FCA. For instance, [6] proposes a parallelization of Ganter's algorithm by decomposing the set of all concepts into non-overlapping subsets which are computed simultaneously. Another parallelization of Ganter's algorithm is presented in [2]. The basic idea in [2] is that the lexicographically ordered power

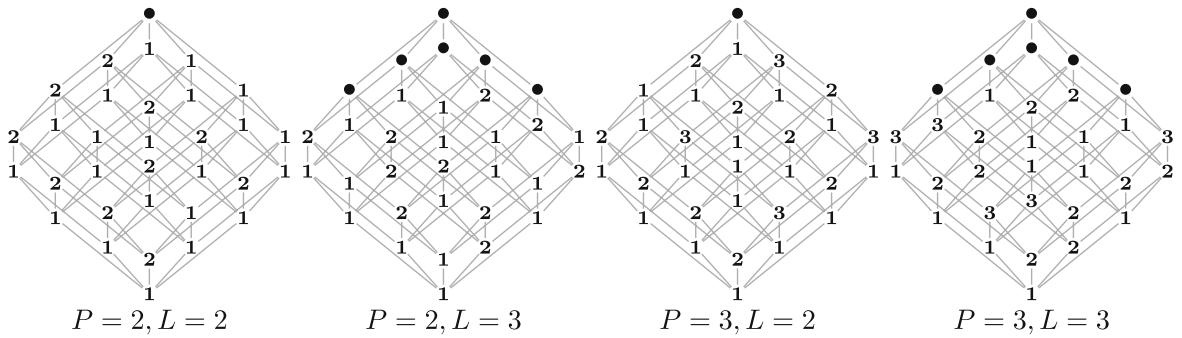

Fig. 3 Examples of parallelization for various values of P and L

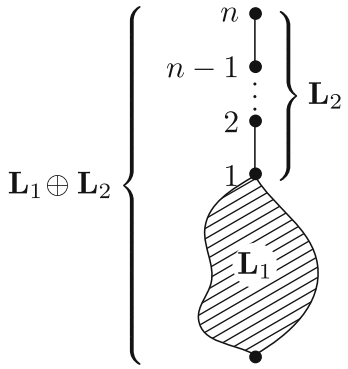

Fig. 4 Ordinal sum $\mathbf{L}_1 \oplus \mathbf{L}_2$ of a lattice $\mathbf{L}_1$ and an n-element chain $\mathbf{L}_2$

set 2^Y is split into p intervals of the same length (p indicates a number of processes). Then, each of the p intervals is executed by an independent process using a serial version of Ganter's algorithm. A different approach is shown, e.g., in [12] where the algorithm is based on dividing the input data into disjoint fragments which are then computed by independent processes. A detailed comparison of the algorithms in terms of their efficiency and scalability is beyond the scope of this paper and will be a subject of future investigation.

4 Efficiency and implementation issues

From the point of view of the worst-case complexity, PCbO is a polynomial time delay [11] algorithm with asymptotic complexity $O(|\mathcal{B}|\cdot|Y|^2\cdot|X|)$ because in the worst case, PCbO can degenerate into the sequential CbO [14, 15]. The actual performance compared to CbO is influenced by the number of processors P and their utilization. In case of optimal utilization of processors, PCbO can run P times faster than CbO, i.e. the reciprocal P^{-1} can be seen as a multiplicative constant of the running time of CbO. In practice, the multiplicative constant is greater than P^{-1} because (i) concepts are not distributed over the processors uniformly and (ii) the parallelization has certain overhead. In order to show how PCbO behaves on average data, we should provide theoretical and experimental average-case complexity analysis. The theoretical analysis seems to be an interesting and challenging problem which is

Table 1 Performance for selected datasets (real time, in seconds; time in parentheses represents total processor time used by all the processors together)

Dataset	Mushroom	Tic-tac-toe	Debian tags	Anon. web
Size	8,124 × 119	958 × 29	14,315 × 475	32,710 × 295
Density	19 %	34 %	< 1 %	1 %
PCbO ($P = 1$)	4.89	0.06	7.79	40.32
PCbO ($P = 2$)	2.78 (5.16)	0.04 (0.07)	5.52 (9.34)	22.16 (43.33)
PCbO ($P = 4$)	1.90 (5.39)	0.03 (0.07)	3.65 (10.88)	13.38 (47.81)
PCbO ($P = 8$)	1.18 (5.58)	0.02 (0.07)	2.51 (11.08)	8.09 (46.68)
Ganter's	834.40	2.15	1,720.82	10,039.73
Lindig's	5,271.98	14.53	2,639.67	13,422.64
Berry's	934.50	5.78	1,531.94	3,615.07

Table 2 Utilization of processors (number of concepts processed by particular processors)

CPU	#0	#1	#2	#3	#4	#5	#6
Mushroom ($P = 2$)	440	103,005	135,265				
Mushroom ($P = 4$)	440	78,825	89,174	24,180	46,091		
Mushroom ($P = 6$)	440	35,486	78,348	23,040	33,398	44,479	23,519
Tic-tac-toe ($P = 2$)	409	31,986	27,110				
Tic-tac-toe ($P = 4$)	409	16,518	13,832	15,468	13,278		
Tic-tac-toe ($P = 6$)	409	11,407	9,962	10,635	7,759	9,944	9,389

yet to be explored. In the sequel we present results of experiments with randomly generated and real data sets which may give hint how PCbO behaves for different values of P and L.

We first compare PCbO with other algorithms [16] for computing formal concepts. Namely, we compare it with Ganter's [7], Lindig's [17] and Berry's [4] algorithms (all implemented in ANSI C). The comparison is made using datasets from [1, 10] and a dataset generated from package descriptions in Debian GNU/Linux. The results, along with the information on sizes and densities (percentage of 1s) of used data sets, are depicted in Table 1. The first four rows contain running times of PCbO that has been run on 1 (sequential version), 2, 4, and 8 hardware processors. The measurements have been done on an otherwise idle 64-bit x86_64 hardware with 8 independent processors (2× Quad-Core Intel Xeon E5345, 2.33 GHz, 12 GB RAM). For $P > 1$, the table in Table 1 contains total processor time used to compute all formal concepts (the time written in parentheses). This time allows us to make a rough estimate of the overhead that is needed to manage multiple threads of computation: the overhead can be computed as the real processor time minus the total processor time divided by P. As it is expected, larger values of P lead to a larger overhead. The utilization of processors can be observed from the number of concepts that are processed by each processor. For instance, Table 2 shows the distribution of computed concepts among particular processors. The processor marked #0, is the initial sequential stage of the algorithm. It should be mentioned that the number of computed concepts by each processor is entirely given by parameters P, L, and by the context. This means, if one processor completes its computation, it cannot "help" other processors to process their load.

The next experiment focuses on the scalability of PCbO, i.e., the ability to decrease the running time using multiple processors. For this set of experiments we have used

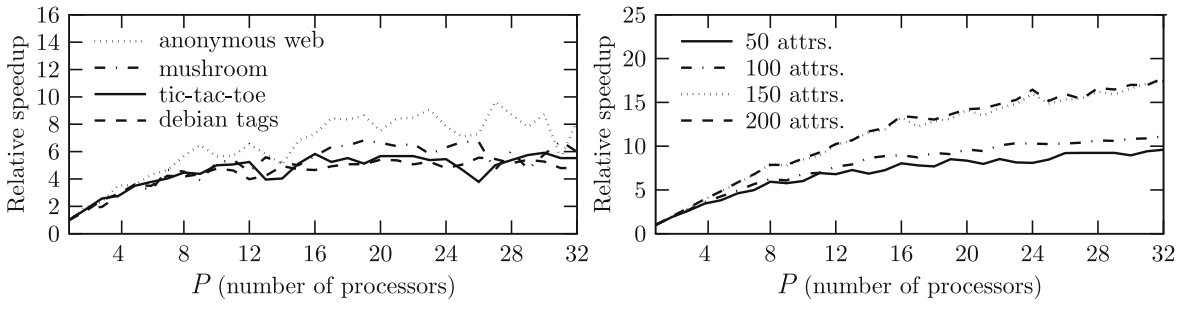

Fig. 5 Relative speedup in various data tables (on the *left*); relative speedup in contexts with various counts of attributes (on the *right*)

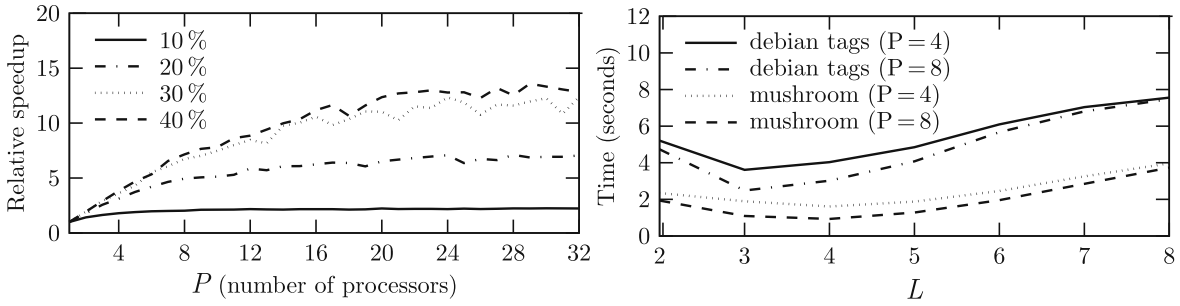

Fig. 6 Relative speedup dependent on density of 1's (on the *left*); running time dependent on the argument L (on the *right*)

computer equipped with eight core UltraSPARC T1 processor that is able to process up to 32 simultaneously running threads. Fig. 5 (left) contains results for selected datasets while Fig. 5 (right) contains results for randomly generated tables with 10,000 objects and 5 % density [16] of 1's. By a *relative speedup* which is shown on the y-axes in the graphs, we mean the theoretical speedup given by the number of hardware processors (e.g., if we have 4 processors, the execution can be 4 times faster). Therefore, the relative speedup is a ratio of running time using a single processor (the sequential algorithm) and running time using multiple processors. Note that the theoretical maximum of the speedup is equal to P but the real speedup is always smaller due to the overhead caused by managing of multiple threads (cf. also Table 1). The experiment in Fig. 6 (left) shows results of the impact of the data density. That is, we have generated data tables with various densities of 1's and observed the impact on the scalability. We have used data tables of size $5{,}000 \times 100$. Finally, Fig. 6 (right) illustrates the influence of parameter L on various data tables and amounts of processors. The experiments indicate that good choice is $L \in \{3, 4\}$, see Remark 5.

Let us note that the actual performance of an implementation of the algorithm depends on used data structures. We have used boolean vectors as basic data structures which turned out to be very efficient. The data structures and optimized algorithms for computing closures are further discussed in Outrata and Vychodil (submitted for publication).

5 Conclusions

We have introduced a parallel algorithm called PCbO for computing formal concepts in object-attribute data tables. The parallel algorithm results as a parallelization of CbO [14, 15] and is formalized by a recursive procedure which simulates the ordinary CbO up to a point where it forks into multiple processes and each process computes a disjoint set of formal concepts. The algorithm has minimal overhead because the concurrent processes computing disjoint sets of concepts are fully independent. This significantly improves efficiency of the algorithm. We have shown that the algorithm is scalable. With growing numbers of CPUs, the speedup of the computation given by increasing number of CPUs is near its theoretical limit. The implementation of the algorithm can be downloaded from

http://fcalgs.sourceforge.net/pcbo-amai.html.

The future research will focus on

- refinements of the algorithm including new approaches to reducing the number of concepts which are computed multiple times, some advances towards this direction can be found in Outrata and Vychodil (submitted for publication);
- comparison of various strategies for selecting queues and advanced conditions preventing degenerate computation, see Remark 6;
- performance comparison with other parallel algorithms, performance and scalability tests of various data structures for representing contexts, extents, and intents;
- specialized variants of the algorithm focused to solve particular problems related to FCA, e.g., factorization of binary matrices [3].

References

1. Asuncion, A., Newman, D: UCI Machine learning repository. School of Information and Computer Sciences, University of California, Irvine (2007)
2. Baklouti, F., Levy G.: A distributed version of the Ganter algorithm for general Galois Lattices. In: Belohlavek, R., Snasel, V. (eds.) Proc. CLA, pp. 207–221 (2005)
3. Belohlavek, R., Vychodil V.: Discovery of optimal factors in binary data via a novel method of matrix decomposition. J. Comput. Syst. Sci. **76**, 3–20 (2010)
4. Berry, A., Bordat, J.-P., Sigayret, A.: A local approach to concept generation. Ann. Math. Artif. Intell. **49**, 117–136 (2007)
5. Carpineto, C., Romano, G.: Concept Data Analysis. Theory and Applications. Wiley, New York (2004)
6. Fu, H., Mephu Nguifo, E.: A Parallel Algorithm to Generate Formal Concepts for Large Data. ICFCA, LNCS **2961**, 394–401 (2004)
7. Ganter, B.: Two basic algorithms in concept analysis. (Technical Report FB4-Preprint No. 831). TH Darmstadt (1984)
8. Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations. Springer, Berlin (1999)
9. Grätzer G. et al.: General Lattice Theory, 2nd edn. Birkhäuser, Basel (2003)
10. Hettich, S., Bay, S.D.: The UCI KDD Archive University of California, Irvine, School of Information and Computer Sciences (1999)
11. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. **27**(3), 119–123 (1988)
12. Kengue J. F. D., Valtchev P., Djamégni C. T.: A parallel algorithm for lattice construction. ICFCA, LNCS **3403**, 249–264 (2005)
13. Kuznetsov, S.: Interpretation on graphs and complexity characteristics of a search for specific patterns. Autom. Doc. Math. Linguist. **24**(1), 37–45 (1989)
14. Kuznetsov, S.: A fast algorithm for computing all intersections of objects in a finite semi-lattice (Быстрый алгоритм построения всех пересечений обIектов из конечной полурешетки, in Russian). Automatic Documentation and Mathematical Linguistics, **27**(5), 11–21 (1993)
15. Kuznetsov, S.: Learning of simple conceptual graphs from positive and negative examples. PKDD, pp. 384–391 (1999)
16. Kuznetsov, S., Obiedkov, S.: Comparing performance of algorithms for generating concept lattices. J. Exp. Theor. Artif. Int. **14**, 189–216 (2002)
17. Lindig, C.: Fast concept analysis. Working with Conceptual Structures—Contributions to ICCS 2000, pp. 152–161. Shaker, Aachen (2000)
18. Norris, E.M.: An Algorithm for computing the maximal rectangles in a binary relation. Rev. Roum. Math. Pures. Appl. **23**(2), 243–250 (1978)
19. Wille, R.: Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. Ordered Sets, pp. 445–470, Reidel, Dordrecht (1982)

Advances in algorithms based on CbO

Petr Krajca, Jan Outrata, Vilem Vychodil*

Department of Computer Science, Palacky University, Olomouc, Czech Republic
Tř. 17. listopadu 12, 771 46 Olomouc, Czech Republic
krajcap@inf.upol.cz,{jan.outrata,vilem.vychodil}@upol.cz

Abstract. The paper presents a survey of recent advances in algorithms for computing all formal concepts in a given formal context which result as modifications or extensions of CbO. First, we present an extension of CbO, so called FCbO, and an improved canonicity test that significantly reduces the number of formal concepts which are computed multiple times. Second, we outline a parallel version of the proposed algorithm and discuss various scheduling strategies and their impact on the overall performance and scalability of the algorithm. Third, we discuss important data preprocessing issues and their influence on the algorithms. Namely, we focus on the role of attribute permutations and present experimental observations about the efficiency of the proposed algorithms with respect to the number of inversions in such permutations.

1 Introduction

The major issue of widely-used algorithms for computing formal concepts, including CbO [12–14], NextClosure [5, 6], or UpperNeighbor [16], is that some concepts are computed multiple times which brings significant overhead. This paper deals with various ways to reduce the overhead. Notice that recently an increasing attention has been paid to various modifications of CbO, see [1, 10, 17].

This paper presents a survey of recent advances in three interconnected areas. First, we present an algorithm called FCbO which achieves better performance than CbO by reducing the total number of formal concepts that are computed multiple times. The reduction is achieved by introducing an additional canonicity test which effectively prunes the CbO tree during the computation. Second, we elaborate on issues related to parallel execution of FCbO. We have already proposed a parallel variant of CbO, so-called PCbO [10]. In this paper, we propose an analogous parallelization of FCbO and we discuss various workload distribution strategies that may have impact on the overall performance of such parallelization. Third, we focus on data preprocessing—an important issue that is often underestimated. Namely, some algorithms for FCA (including those from the CbO family) achieve better performance if attributes are processed in particular order. In this paper, we present a preliminary study of the role of attribute permutations on the performance of CbO and the derived algorithms.

* Supported by grant no. P103/10/1056 of the Czech Science Foundation and by grant no. MSM 6198959214.

Notation Throughout the paper, $X = \{0, 1, \ldots, m\}$ and $Y = \{0, 1, \ldots, n\}$ are finite nonempty sets of objects and attributes, respectively, and $I \subseteq X \times Y$ is an incidence relation. The triplet $\langle X, Y, I \rangle$ is a formal context. The concept-forming operators induced by I will be denoted by $^{\uparrow_I} : 2^X \to 2^Y$ and $^{\downarrow_I} : 2^Y \to 2^X$, respectively, see [6] for details. We assume that reader has knowledge of basic algorithms for FCA.

2 FCbO: Fast Close-by-One with New Canonicity Test

In this section we briefly describe the new canonicity test and a new algorithm derived from CbO which uses this test. Recall that the original canonicity test used by CbO (and NextClosure) is always used *after* a new formal concept is computed. For $B \subseteq Y$ and $j \notin B$, one checks whether

$$B \cap Y_j = D \cap Y_j, \text{ where } D = (B \cup \{j\})^{\downarrow_I \uparrow_I} \tag{1}$$

and $Y_j = \{y \in Y \mid y < j\}$. FCbO employs an additional test that is performed *before* D is computed, eliminating thus the computation of $^{\downarrow_I \uparrow_I}$. Notice that (1) fails iff $B \oslash j \neq \emptyset$, where

$$B \oslash j = (D \setminus B) \cap Y_j = \big((B \cup \{j\})^{\downarrow_I \uparrow_I} \setminus B\big) \cap Y_j. \tag{2}$$

The new canonicity test exploits the fact that if (1) fails for given B and $j \notin B$, the monotony of $^{\downarrow_I \uparrow_I}$ yields that the test will also fail for each $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \not\subseteq B'$. The conclusion can be done without computing D. If $B \oslash j \subseteq B'$, we are still compelled to perform the original canonicity test. Thus, the new (additional) canonicity test is based on the following assertion:

Lemma 1 (See [17]). *Let $B \subseteq Y$, $j \notin B$, and $B \oslash j \neq \emptyset$. Then, for each $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \not\subseteq B'$, we have $B' \oslash j \neq \emptyset$.* □

FCbO can be seen as an extended version of CbO in that we propagate the information about sets (2) which take part in the new test. The information is propagated in the top-down direction. In order to apply the new test, we have to change the search strategy of the algorithm from the depth-first search (as it is in CbO) to a combined depth-first and breadth-first search. FCbO is represented by a recursive procedure FastGenerateFrom, see Algorithm 1, which accepts three arguments: a formal concept $\langle A, B \rangle$ (an initial formal concept), an attribute $y \in Y$ (first attribute to be processed), and a set $\{N_y \subseteq Y \mid y \in Y\}$ of subsets of attributes Y the purpose of which is to carry information about sets (2). Each invocation of FastGenerateFrom has its own local *queue* used to store information about computed concepts. Unlike CbO, if the canonicity tests succeed (line 7 and line 10), we do not call FastGenerateFrom recursively but we store information about the concept in the queue (line 11). After each attribute is processed, we perform the recursive calls, see lines 17–19. The new canonicity test is performed in line 7 based on information stored in N_j's. The original canonicity test is performed in line 10. If the test in line 10 fails, we update the contents of M_j, see line 13. Note that M_j's can be seen as local copies

Algorithm 1: Procedure FastGenerateFrom($\langle A, B\rangle, y, \{N_y \mid y \in Y\}$)

```
1  list ⟨A, B⟩        // concept ⟨A, B⟩ is processed, e.g., listed or stored
   // check halting condition of the current call
2  if B = Y or y > n then
3  |  return
4  end
   // process all attributes beginning with y
5  for j from y upto n do
6  |  set M_j to N_j                                  // M_j is a pointer to N_j
   |  // perform new canonicity test
7  |  if j ∉ B and N_j ∩ Y_j ⊆ B ∩ Y_j then
   |  |  // compute new concept ⟨C, D⟩ = ⟨A ∩ {j}^↓I, (B ∪ {j})^↓I↑I⟩
8  |  |  set C to A ∩ {j}^↓I
9  |  |  set D to C^↑I
   |  |  // perform original canonicity test
10 |  |  if B ∩ Y_j = D ∩ Y_j then
   |  |  |  // store new concept for further processing
11 |  |  |  put ⟨⟨C, D⟩, j⟩ to queue
12 |  |  else
   |  |  |  // update information about implied attributes
13 |  |  |  set M_j to D                    // M_j becomes a pointer to D
14 |  |  end
15 |  end
16 end
   // perform recursive calls of FastGenerateFrom
17 while get ⟨⟨C, D⟩, j⟩ from queue do
18 |  FastGenerateFrom(⟨C, D⟩, j + 1, {M_y | y ∈ Y})
19 end
   // terminate current call
20 return
```

of N_j's which are used as the third argument for consecutive calls of FastGenerateFrom. Sets N_j are used instead of (2) because it is actually easier (and more efficient) to maintain a set of pointers to intents than to compute (and allocate memory for) sets (2) during the computation.

FCbO is correct: when invoked with $\langle \emptyset^{\downarrow_I}, \emptyset^{\downarrow_I\uparrow_I}\rangle$, $y = 0$, and $\{N_y = \emptyset \mid y \in Y\}$, Algorithm 1 lists all formal concepts in $\langle X, Y, I\rangle$ in the same order as CbO, each of them exactly once. Let us note that FCbO can be turned into a "Fast NextClosure" (i.e., an algorithm that lists concepts in the lexicographical order [5]) by either (i) using a *stack* instead of a *queue* or by (ii) modifying the loop in line 5 so that it goes "**from** n **downto** y". See [17] for further details on FCbO.

Example 1. Consider a context with $X = \{0, \ldots, 3\}$, $Y = \{0, \ldots, 5\}$, and $I = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 0,2\rangle, \langle 1,0\rangle, \langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 2,0\rangle, \langle 2,1\rangle, \langle 2,4\rangle, \langle 3,1\rangle, \langle 3,2\rangle\}$. This formal context induces 12 formal concepts denoted $C_1, \ldots, C_{12}$. In case of both CbO and FCbO, the computation can be depicted by a tree. Moreover, a

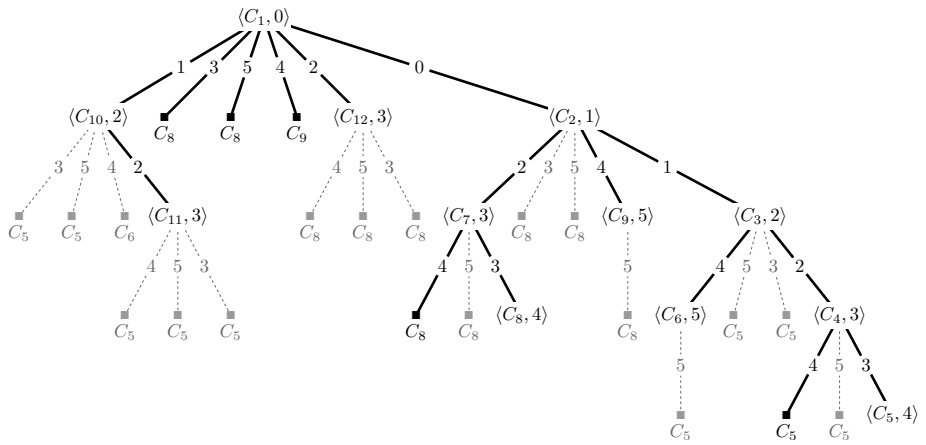

Fig. 1. Example of an FCbO tree—a pruned CbO tree.

	concepts	closures (CbO)	closures (FCbO)	ratio (CbO)	ratio (FCbO)
mushroom	238,710	4,006,498	426,563	5.9 %	55.9 %
anon. web	129,009	27,949,552	1,475,341	0.4 %	8.7 %
debian tags	38,977	12,045,680	679,911	0.3 %	5.7 %
tit-tac-toe	59,505	221,608	128,434	26.8 %	46.3 %

Table 1. Total numbers of closures computed by CbO and FCbO.

FCbO tree is a pruned version of the CbO tree, see Fig. 1. The black-square nodes represent concepts computed multiple times by FCbO and CbO whereas the grey-square nodes represent concepts computed multiple times by CbO and not computed by FCbO. Therefore, grey nodes and edges in Fig. 1 denote subtrees pruned using the new canonicity test. In this case, the number of concept computed multiple times is significantly reduced.

Experimental Evaluation We have evaluated FCbO and compared CbO and FCbO using various real data sets and artificial data sets. The impact of the new canonicity test is presented in Table 1 comparing the total numbers of closures computed by CbO and FCbO in selected benchmark data sets [2, 8]. The table includes numbers of concepts and ratios of the number of computed closures to the number of (distinct) formal concepts in the data, i.e., the frequency of successful canonicity tests. Apparently, FCbO has a higher rate of successful canonicity tests than CbO. Thus, in terms of the number of computed closures, FCbO is more efficient than CbO. Since the total number of computed closures directly influences the speed of the algorithm, FCbO is (usually) faster than CbO [17]. The reduction of total time needed for computing all formal concepts is apparent from Table 2. The table shows total time (in seconds) needed to

	mushroom	tic-tac-toe	debian tags	anon. web
size	$8,124 \times 119$	958×29	$14,315 \times 475$	$32,710 \times 295$
density	19 %	34 %	< 1 %	1 %
FCbO	0.23	0.02	0.10	0.15
CbO	4.34	0.06	5.31	27.14
NextClosure	685.00	1.86	1,432.25	8,236.85
UpperNeighbor	4,368.19	12.54	2,159.80	11,068.52
Berry's [3]	950.73	6.93	1,512.73	4,421.51

Table 2. Performace of algorithms (speed in seconds).

analyze the data sets. For the purpose of comparison the table contains also other well-known algorithms. The experiments were performed on an Apple MacPro computer equipped with two quad-core processors (Intel Xeon, 2.8 GHz) and 16 GB of RAM and all algorithms were implemented in ANSI C using bitarray representation [11]. Notice that in the worst case, FCbO collapses into CbO (e.g., in case of I being the inequality relation on $X = Y$). FCbO is a polynomial time-delay algorithm [7, 9] because the additional canonicity test has a linear time-delay overhead compared to CbO, see [17] for further details on FCbO and its performance.

3 PFCbO: Parallel FCbO and Workload Distribution

This section is devoted to parallelization issues of FCbO. Recall that in [10], we have described PCbO which results by a parallelization of CbO. FCbO can be turned into a parallel algorithm in much the same way as the original CbO can be turned into PCbO. Since the procedure of parallelization is fairly similar to that presented in [10], we focus mainly on issues that are not discussed in [10]. Namely, we compare several strategies to balance the workload distribution among independent processors and compare their efficiency.

Following the ideas from [10], a parallel variant of FCbO consists of three stages: First, we compute and process all concepts that are derivable in less than L steps. Second, we store all concepts derivable in exactly L steps in a new *queue*. Third, we distribute concepts from the *queue* among P independent processors and we let each of the processors compute the remaining concepts using FCbO. Typically, each processor r has its own queue denoted $queue_r$ containing concepts assigned to this processor. A parallel algorithm based on these ideas shall be called *Parallel Fast Close-byOne* (PFCbO).

Clearly, the practical efficiency of both PCbO and PFCbO depends on the choice of the strategy that distributes concepts among processors during the third step of the computation. The decision how to assign concepts to particular queue is generally difficult since we do not know the distribution of formal concepts in the search space of all formal concepts until we actually compute them all and reveal the structure of the call tree. As a consequence, the distribution

of workload may be in some cases unbalanced. In [10], we have used a simple round-robin principle which turned out to be reasonably efficient. Nevertheless, there are other schemes of the workload distribution that can be considered:

(i) *round-robin*—concepts are distributed to queues attached to each processor, in the way that n-th concept is placed into a $queue_r$ where $r = (n \bmod P)+1$ and P is the number of processors. For instance, if we consider $P = 4$ and concepts $C_1, \ldots, C_{10}$, they are assigned to queues as follows:

$$queue_1 = \{C_1, C_5, C_9\}, \qquad queue_2 = \{C_2, C_6, C_{10}\},$$
$$queue_3 = \{C_3, C_7\}, \qquad queue_4 = \{C_4, C_8\}.$$

(ii) *zig-zag*—this strategy is similar to the previous strategy but it uses a different formula to determine the $queue_r$. The $queue_r$ is given by

$$r = \min\big(n \bmod z, z - (n \bmod z)\big) + 1 \tag{3}$$

where $z = 2 \times P + 1$ assuming that P is number of processors. For $P = 4$ and concepts $C_1, \ldots, C_{10}$ the distribution of concepts is

$$queue_1 = \{C_1, C_8, C_9\}, \qquad queue_2 = \{C_2, C_7, C_{10}\},$$
$$queue_3 = \{C_3, C_6\}, \qquad queue_4 = \{C_4, C_5\}.$$

(iii) *blocks*—this workload distribution scheme divides the queue of all concepts into chunks of approximately equal size and these "blocks of concepts" are redistributed into the queues of independent processors. In this case, the n-th concept is placed into $queue_r$, where

$$r = \left\lceil \frac{(n \times P)}{Q} \right\rceil \tag{4}$$

with P being the number of processors, Q being the number of all concepts, and $\lceil x \rceil$ being the usual ceiling function. For instance, in case of $C_1, \ldots, C_{10}$ (i.e., $Q = 10$) and four queues (i.e., $P = 4$), we get:

$$queue_1 = \{C_1, C_2\}, \qquad queue_2 = \{C_3, C_4, C_5\},$$
$$queue_3 = \{C_6, C_7\}, \qquad queue_4 = \{C_8, C_9, C_{10}\}.$$

(iv) *fair*—all concepts remain stored in one shared queue and each processor gets concepts from the queue one by one. The benefit of this scheme is that it allows to react on the revealing structure of the call tree. On the other hand, this method of distributing concepts requires synchronization among processors while accessing this queue. Note that in contrast to the above-described schemes, this scheme has no fixed structure and the workload is distributed non-deterministically.

(v) *random*—the workload is spread among processors randomly. We are considering this strategy to be referential and it is included for the purpose of comparison.

Experimental Evaluation In order to evaluate the strategies of workload distribution, we have tested our algorithm for each strategy using various data sets and various number of processors. Table 3 depicts the time needed to compute all formal concepts using particular strategy. Surprisingly, there are only small

	round-r.	*blocks*	*fair*	*zig-zag*	*random*
debian tags	0.0974	0.0988	0.0938	0.0984	0.0986
anon. web	0.1518	0.1590	0.1500	0.1528	0.1522
mushroom	0.1772	0.2158	0.1550	0.1788	0.1820
tic-tac-toe	0.0172	0.0198	0.0168	0.0174	0.0180
random ($5000 \times 100 \times 10$)	0.0806	0.1194	0.0796	0.0820	0.0876
random ($10000 \times 100 \times 15$)	1.1380	2.1326	0.8698	1.0974	1.1670

Table 3. Performace under various workload distributions (speed in seconds).

differences among the considered schemes of the workload distribution, i.e., the ordinary round-robin used in [10] is indeed adequate for the job. Nevertheless, the *fair* strategy seems to be the most efficient. One can see that the *round-robin* and *zig-zag* strategies provide performance slightly better than the random workload distribution. On the other hand, the *blocks* scheme of distribution provides performance even worse than the random distribution and seems to be inappropriate for PFCbO.

4 Data Preprocessing Issues

Algorithms for computing concepts can be classified in many ways, see, e.g. [15]. An important attribute of algorithms for FCA is whether their performance depends on the order of objects and attributes in the input data table. Therefore, an algorithm for computing formal concepts shall be called (*permutation*) *resistant* whenever all isomorphic copies of a formal context $\langle X, Y, I\rangle$ with $Y = \{0, 1, \ldots, n\}$ require the same number of elementary computation steps in order to compute all concepts. For our purposes, an elementary computation step will be represented by computation of a single fixpoint of the concept-forming operators $^{\uparrow_I}$ and $^{\downarrow_I}$.

One can easily see that, e.g., Lindig's UpperNeighbor algorithm [16] is resistant. On the other hand, CbO and FCbO are not resistant. Indeed, a different order of attributes in a data table can yield different CbO and FCbO trees that may have different numbers of nodes (notice that the loop in line 5 of Algorithm 1 processes attributes from left to right). Since CbO and FCbO are not resistant, a proper ordering of attributes before computation can further reduce the number of concepts that are computed multiple times, thus improving the efficiency. In this section, we investigate particular permutations of attributes and explore the impact of inversions on the number of computed closures.

In order to describe various formal contexts with respect to the structure of the data table, we introduce a notion of an ordered formal context and inversion:

Definition 1. *An ordered formal context is a formal context* $\langle X, Y, I\rangle$ *where* $Y = \{0, \ldots, n\}$ *and for all attributes*

$$|\{0\}^{\downarrow_I}| \leq |\{1\}^{\downarrow_I}| \leq \cdots \leq |\{n\}^{\downarrow_I}|. \tag{5}$$

A pair of attributes $\langle y_1, y_2 \rangle \in Y \times Y$ *such that* $|\{y_1\}^{\downarrow_I}| \not\leq |\{y_2\}^{\downarrow_I}|$ *shall be called an inversion.*

Verbally, the attributes in an ordered formal context are sorted in the ascending order according to their support, i.e., the number of objects having these attributes. As a consequence of the previous definition, an ordered formal context contains no inversions.

From the point of view of formal concepts and concept lattices, the order of objects and attributes in which they appear in the data table is not essential. Therefore, one can reorder attributes in an arbitrary way. From the computational point of view, however, it may happen that certain orderings of attributes yield better results in conjunction with particular argorithms than other orderings. In case of our algorithms, the order has an important impact on the process of the execution of both CbO and FCbO since the canonicity test is driven by the order of attributes. The following assertions show that for an ordered formal context with pairwise distinct columns, the canonicity tests succeed for all attribute concepts. We first prove a technical claim:

Lemma 2. *Let* $\langle X, Y, I \rangle$ *be an ordered formal context with* $Y = \{0, \ldots, n\}$. *Then, for each* $k, j \in Y$ *such that* $k < j$, *we have* $k \in \{j\}^{\downarrow_I \uparrow_I}$ *iff* $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$.

Proof. Note that attributes from Y are integers and "$<$" denotes the usual strict linear order on the set of all integers. Suppose that $k \in \{j\}^{\downarrow_I \uparrow_I}$, i.e., $\{k\} \subseteq \{j\}^{\downarrow_I \uparrow_I}$. By the antitony of $^{\downarrow_I}$, we get $\{k\}^{\downarrow_I} \supseteq \{j\}^{\downarrow_I \uparrow_I \downarrow_I} = \{j\}^{\downarrow_I}$. Thus, it remains to show the converse inclusion. Since $\langle X, Y, I \rangle$ is ordered and $k < j$, we get $|\{k\}^{\downarrow_I}| \leq |\{j\}^{\downarrow_I}|$, see (5). Hence, $|\{k\}^{\downarrow_I}| \leq |\{j\}^{\downarrow_I}|$ and $\{k\}^{\downarrow_I} \supseteq \{j\}^{\downarrow_I}$ yield $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$. Conversely, if $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$ then obviously $k \in \{j\}^{\downarrow_I \uparrow_I}$, proving the claim. □

Applying Lemma 2, we get:

Theorem 1. *Let* $\langle X, Y, I \rangle$ *be an ordered formal context where* $Y = \{0, \ldots, n\}$ *and* $\{a\}^{\downarrow_I} \neq \{b\}^{\downarrow_I}$ *for any* $a, b \in Y$. *Then for each* $j \in Y$ *such that* $j \notin \emptyset^{\downarrow_I \uparrow_I}$,

$$\emptyset^{\downarrow_I \uparrow_I} \cap Y_j = \{j\}^{\downarrow_I \uparrow_I} \cap Y_j, \tag{6}$$

where $Y_j = \{y \in Y \,|\, y < j\}$.

Proof. Take $j \in Y$ such that $j \notin \emptyset^{\downarrow_I \uparrow_I}$. Observe that condition (6) holds true iff there is no attribute $k \in Y$ such that $k \notin \emptyset^{\downarrow_I \uparrow_I}$, $k < j$, and $k \in \{j\}^{\downarrow_I \uparrow_I}$. Thus, consider any $k \in Y$ such that $k < j$. Since $\langle X, Y, I \rangle$ is ordered, our assumption $j \notin \emptyset^{\downarrow_I \uparrow_I}$ yields $k \notin \emptyset^{\downarrow_I \uparrow_I}$. By the assumption, $\{k\}^{\downarrow_I} \neq \{j\}^{\downarrow_I}$, i.e., Lemma 2 yields $k \notin \{j\}^{\downarrow_I \uparrow_I}$, finishing the proof. □

Theorem 1 shows that for an ordered formal context with pairwise distinct columns, invocations of FastGenerateFrom in the first level of recursion always succeeds and generates concepts. Moreover, from the proof of Theorem 1 it follows that in any ordered formal context, the first derivation [10] does not exists for attribute j if there is an attribute k such that $k < j$ and $\{k\}^{\downarrow_I} = \{j\}^{\downarrow_I}$.

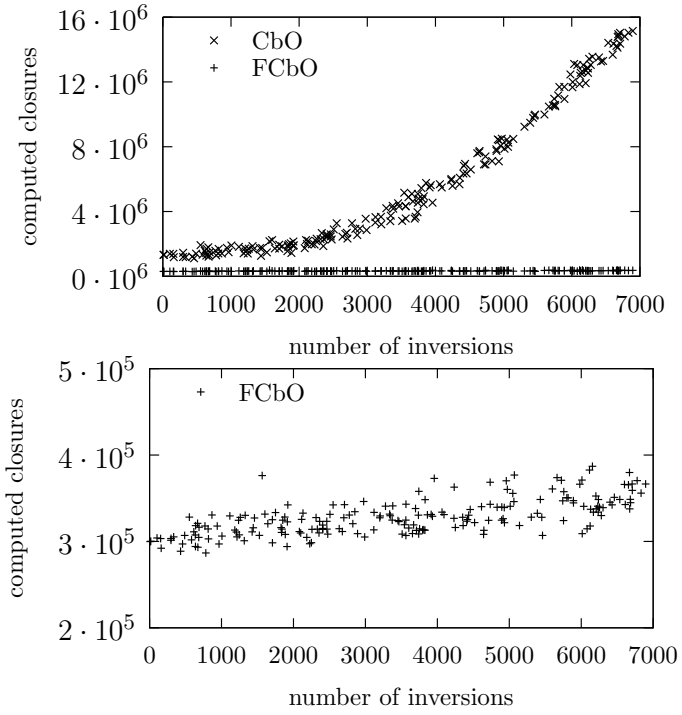

Fig. 2. Impact of inversion in the mushrooms data set

This has a practical consequence for the parallel variants of CbO and FCbO in case of ordered contexts, because it allows us to determine the number of concepts generated during the first stages of the algorithms. If the number of attributes is significantly larger than the number of processors, and this condition is usually fulfilled, it is sufficient to compute only the first derivations and then distribute the workload among all processors.

Furthermore, our empirical experiments have shown an interesting tendency that while processing ordered formal contexts, canonicity tests fail less frequently than in case of contexts containing inversions. In addition, the experiments have shown that with the increasing number of inversions in a data table, the average number of computed closures grows. For instance, Fig. 2 shows how the number of inversions in the *mushroom* data set affects the total number of computed closures. The first graph (at the top) depicts this dependency for CbO and FCbO. The second graph (at the bottom) provides a more detailed view for FCbO. Similar tendency can be observed for other benchmark data sets.

Remark 1. Let us note that the ordering of attributes introduced by (5) has already been used in [4] but the purpose of the ordering in [4] is different. In [4], the authors use this particular ordering of attributes in a parallel version of Ganter's NextClosure algorithm to achieve soundness of the algorithm (each

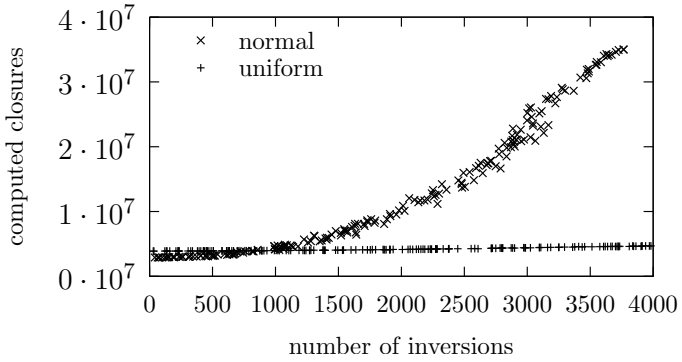

Fig. 3. Impact of inversions in two artificial data sets

	concepts	closures (unordered)	closures (ordered)	ratio (unordered)	ratio (ordered)
mushroom	238,710	426,563	299,201	55.9 %	79.7 %
anon. web	129,009	1,475,341	398,147	8.7 %	32.4 %
debian tags	38,977	679,911	298,641	5.7 %	13.0 %
tit-tac-toe	59,505	128,434	89,930	46.3 %	66.1 %

Table 4. Numbers of computed closures in case of (un)ordered attributes.

concept is then listed only once) while in our case, [4] is used for the sake of increased efficiency.

Remark 2. We have observed a general tendency that certain data sets are more affected by the above-discussed phenomenon than others. For instance, if 1's in a data table are approximately uniformly spread among attributes (i.e., each attribute has approximately the same support), the ordering of attributes (usually) does not have a considerable effect on decreasing the number of closures. Fig. 3 depicts how the increasing number of inversions affects the number of computed closures in two artificial data sets where 1's are distributed (i) approximately uniformly among the attributes and (ii) approximately normally among the attributes. Both data sets have the same parameters in that they consist of 1000 objects, 100 attributes, and contain 15 % of 1's, however, the distributions of 1's among the attributes are quite different. As one can see from Fig. 3, the impact of the number of inversions on the number of computed closures is more significant in case of normally distributed 1's among the attributes.

Experimental Evaluation From our observations it follows that it is desirable to incorporate a preprocessing step which transforms a formal context into a corresponding ordered formal context. In order to evaluate the benefits of this preprocessing step, we have used similar approach as in case of evaluation of

	mushroom	tic-tac-toe	debian tags	anon. web
PFCbO ($P = 1$)	0.23	0.02	0.10	0.15
PFCbO ($P = 2$)	0.14	0.01	0.07	0.11
PFCbO ($P = 4$)	0.09	0.01	0.06	0.09
PFCbO ($P = 8$)	0.06	0.01	0.06	0.08
PCbO ($P = 1$)	4.34	0.06	5.31	27.14
PCbO ($P = 2$)	2.39	0.03	3.59	14.77
PCbO ($P = 4$)	1.65	0.02	2.59	9.22
PCbO ($P = 8$)	0.99	0.01	1.85	5.60

Table 5. Performace with multiple processors (speed in seconds).

the new canonicity test. We have focused on the total numbers of closures and concepts computed by FCbO while processing various ordered and unordered data sets. The results are presented in Table 4 which also includes corresponding ratios.

Apparently, reordering of attributes reduces the number of computed closures, and thus, can reduce time of computation. Note that the Ganter's algorithm [5, 6] is in principle equivalent to CbO. As such, it is also not permutation resistant. Thus, the preprocessing step which reorders attributes can also increase its performance.

5 Overall Evaluation

So far, we have proposed and evaluated several improvements and refinements of the original CbO and PCbO algorithms, namely, new canonicity test, workload distribution schemes, and reordering attributes. However, we have evaluated the impact of each improvement separately. Therefore, we conclude this paper with the evaluation of PFCbO which includes all these improvements.

Table 5 table shows the total time (in seconds) needed to compute all formal concepts in the benchmark data sets using PCbO and PFCbO run on the Apple MacPro computer, equipped with eight processor cores. The parameter P indicates the number of used processors for particular experiment.

Fig. 4 demonstrates the scalability of PFCbO, i.e., the ability to decrease the time of computation by using more processors. In the depicted two experiments, we have used computer equipped with Sun UltraSPARC T1 processor having eight cores (each capable to process up to 4 threads simultaneously) and 8 GB of RAM. Fig. 4 (at the top) shows relative speed up for data sets having 10000 objects, 10 % density of 1's in the data table and various counts of attributes. Fig. 4 (at the bottom) shows relative speed up for data sets having 1000 objects, 100 attributes, and various densities of 1's in the data tables. The 1's in both data sets spread approximately normally among the attributes. Note that each graph contains a certain point from which the increasing number of processors does not allow to take advantage of more processors and performance of the

algorithm may even decline due to the overhead related to the management of multiple threads of execution. However, this is a quite common behavior of parallel algorithms.

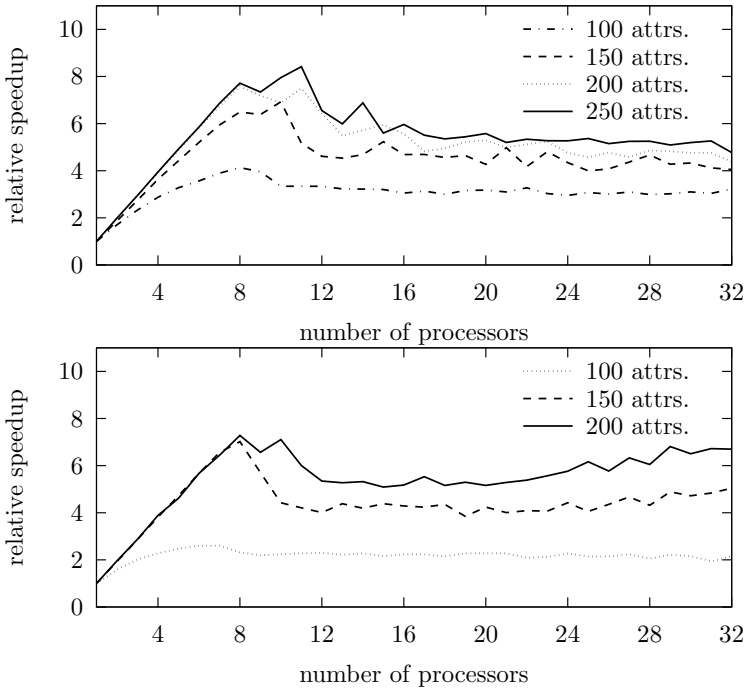

Fig. 4. Scalability of PFCbO

References

1. Andrews S.: In-Close, a Fast Algorithm for Computing Formal Concepts. In: Rudolph, Dau, Kuznetsov (Eds.): *Supplementary Proceedings of ICCS '09*, CEUR WS **483**(2009), 14 pages.
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-483/paper1.pdf
2. Asuncion A., Newman D.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, 2007.
3. Berry A., Bordat J.-P., Sigayret A.: A local approach to concept generation. *Annals of Mathematics and Artificial Intelligence*, **49**(2007), 117–136.
4. Fu H., Mephu Nguifo E.: A Parallel Algorithm to Generate Formal Concepts for Large Data. *ICFCA 2004, LNCS* **2961**, pp. 394–401.
5. Ganter B.: *Two basic algorithms in concept analysis.* (Technical Report FB4-Preprint No. 831). TH Darmstadt, 1984.
6. Ganter B., Wille R.: *Formal concept analysis. Mathematical foundations.* Berlin: Springer, 1999.

7. Goldberg L. A.: *Efficient Algorithms for Listing Combinatorial Structures.* Cambridge University Press, 1993.
8. Hettich S., Bay S. D.: The UCI KDD Archive University of California, Irvine, School of Information and Computer Sciences, 1999.
9. Johnson D. S, Yannakakis M., Papadimitriou C. H.: On generating all maximal independent sets. *Information Processing Letters* **27**(3)(1988), 119–123.
10. Krajca P., Outrata J., Vychodil V.: Parallel Recursive Algorithm for FCA. In: Belohlavek, Kuznetsov (Eds.): *Proc. CLA 2008*, CEUR WS **433**(2008), 71–82. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-433/paper6.pdf
11. Krajca P., Vychodil V.: Comparison of data structures for computing formal concepts. In: *Proc. MDAI 2009, LNAI* **5861**(2009), 114–125.
12. Kuznetsov S.: Interpretation on graphs and complexity characteristics of a search for specific patterns. *Automatic Documentation and Mathematical Linguistics*, **24**(1)(1989), 37–45.
13. Kuznetsov S.: A fast algorithm for computing all intersections of objects in a finite semi-lattice (Быстрый алгоритм построения всех пересечений объектов из конечной полурешетки, in Russian). *Automatic Documentation and Mathematical Linguistics*, **27**(5)(1993), 11–21.
14. Kuznetsov S.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. *PKDD 1999*, pp. 384–391.
15. Kuznetsov S., Obiedkov S.: Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Int.*, **14**(2002), 189–216.
16. Lindig C.: Fast concept analysis. *Working with Conceptual Structures––Contributions to ICCS 2000,* pp. 152–161, 2000. Aachen: Shaker Verlag.
17. Outrata J., Vychodil V.: Fast algorithm for computing fixpoints of galois connections induced by object-attribute relational data (in preparation).

Information Sciences 185 (2012) 114–127

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data

Jan Outrata *, Vilem Vychodil

Dept. Computer Science, Palacky University, 771 46 Olomouc, Czech Republic

ARTICLE INFO

Article history:
Received 8 November 2010
Received in revised form 14 May 2011
Accepted 19 September 2011
Available online 24 September 2011

Keywords:
Galois connection
Object-attribute data
Formal concept analysis
Frequent itemset mining

ABSTRACT

Fixpoints of Galois connections induced by object-attribute data tables represent important patterns that can be found in relational data. Such patterns are used in several data mining disciplines including formal concept analysis, frequent itemset and association rule mining, and Boolean factor analysis. In this paper we propose efficient algorithm for listing all fixpoints of Galois connections induced by object-attribute data. The algorithm, called FCbO, results as a modification of Kuznetsov's CbO in which we use more efficient canonicity test. We describe the algorithm, prove its correctness, discuss efficiency issues, and present an experimental evaluation of its performance and comparison with other algorithms.

1. Introduction and Preliminaries

This paper describes a new algorithm for computing fixpoints of Galois connections. In particular, we focus on Galois connections [5,12,26,33] that appear in formal concept analysis (FCA) – a method of qualitative analysis of object-attribute relational data [10,33]. In a broader sense, the algorithm belongs to an important family of algorithms for listing combinatorial structures [11] and algorithms for biclustering [3,29]. The algorithm we propose is a refinement of Kuznetsov's [19,21] Close-by-One algorithm (CbO) in which we improve the canonicity test. The improvement significantly reduces the number of fixpoints which are computed multiple times, resulting in an algorithm that is considerably faster than the original CbO.

Recall that an antitone Galois connection between nonempty sets X and Y is a pair $\langle f,g\rangle$ of maps $f: 2^X \to 2^Y$ and $g: 2^Y \to 2^X$ satisfying, for any $A, A_1, A_2 \subseteq X$ and $B, B_1, B_2 \subseteq Y$,

$$A \subseteq g(f(A)), \tag{1}$$
$$B \subseteq f(g(B)), \tag{2}$$
$$\text{if } A_1 \subseteq A_2 \text{ then } f(A_2) \subseteq f(A_1), \tag{3}$$
$$\text{if } B_1 \subseteq B_2 \text{ then } g(B_2) \subseteq g(B_1). \tag{4}$$

The composed maps $f \circ g : 2^X \to 2^X$ and $g \circ f : 2^Y \to 2^Y$ are closure operators in 2^X and 2^Y, respectively [10,12]. A pair $\langle A,B\rangle \in 2^X \times 2^Y$ is called a fixpoint of $\langle f,g\rangle$ if $f(A) = B$ and $g(B) = A$. Since we are interested in listing all fixed points of $\langle f,g\rangle$, we restrict ourselves to finite X and Y.

Galois connections appear as induced structures in data analysis. Namely, suppose that X and Y are sets of objects and attributes/features, respectively, and let $I \subseteq X \times Y$ be an incidence relation, $\langle x,y\rangle \in I$ saying that object $x \in X$ has attribute

* Corresponding author.
E-mail addresses: jan.outrata@upol.cz (J. Outrata), vychodil@acm.org (V. Vychodil).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.09.023

$y \in Y$. In FCA, the triplet $\langle X, Y, I\rangle$ is called a formal context and represents the input object-attribute data. Given $I \subseteq X \times Y$, we introduce two concept-forming operators [10] $^{\uparrow_I} : 2^X \to 2^Y$ and $^{\downarrow_I} : 2^Y \to 2^X$ defined, for each $A \subseteq X$ and $B \subseteq Y$, by

$$A^{\uparrow_I} = \{y \in Y \mid \text{for each } x \in A : \langle x, y\rangle \in I\}, \tag{5}$$

$$B^{\downarrow_I} = \{x \in X \mid \text{for each } y \in B : \langle x, y\rangle \in I\}. \tag{6}$$

By definition (5), $A^{\uparrow_I}$ is the set of all attributes shared by all objects from A and, by (6), $B^{\downarrow_I}$ is the set of all objects sharing all attributes from B. It is easily seen that $\langle {}^{\uparrow_I}, {}^{\downarrow_I}\rangle$ is a Galois connection between X and Y and it shall be called a Galois connection induced by I. The fixpoints of $\langle {}^{\uparrow_I}, {}^{\downarrow_I}\rangle$ are called formal concepts in I [10,12]. Formal concepts represent basic patterns that can be found in I and that have two common interpretations: (i) a geometric one: formal concepts are maximal rectangular subsets of I; (ii) a conceptual one: each formal concept $\langle A, B\rangle$ represents a concept in data with an extent A (objects that fall under the concept) and an intent B (attributes covered by the concept) such that A is a set of objects sharing all attributes from B and B is the set of all attributes shared by all objects from A. The latter interpretation of concepts is inspired by a traditional understanding of concepts as notions having their extent and intent which goes back to traditional Port-Royal logic [8,23].

In this paper, we propose an algorithm that lists all formal concepts in I, each of them exactly once. In the past, there have been proposed various algorithms for solving this task, see [22] for a survey and comparison. One of the main issues solved by all the algorithms is how to prevent listing the same formal concept multiple times. There are several approaches to cope with the problem. For instance, Lindig's algorithm [24] stores found concepts in a data structure (a particular search tree) and uses the data structure to check whether a formal concept has already been found. On the other hand, Ganter's NextClosure [9], CbO [19,21], and the algorithm proposed by Norris [30] use canonicity tests: formal concepts are supposed to be listed in certain order. The fact whether two consecutive concepts are listed in the order is ensured by a canonicity test. If a newly computed formal concept does not pass the canonicity test, it is not further considered. Hence, the canonicity test ensures that even if a formal concept is computed several times, it is listed exactly once. Conceptually, our algorithm can be seen as an improved version of CbO [19,21] in which we modify the canonicity test. The improvement significantly reduces the number of formal concepts which are computed multiple times. The reduction has a great impact on the performance of the algorithm because computing formal concepts using the closures $A^{\uparrow_I\downarrow_I}$ or $B^{\downarrow_I\uparrow_I}$ of a set of objects A or a set of attributes B, respectively, is the most critical operation. Note that other promising approaches related to CbO have been introduced in [27] and recently in [2].

Let us stress the importance of listing formal concepts. First, formal concepts are the basic output of formal concept analysis. If we denote by $\mathcal{B}(X, Y, I)$ the set of all formal concepts in $I \subseteq X \times Y$, we can define a partial order $\leqslant$ on $\mathcal{B}(X, Y, I)$ as follows:

$$\langle A_1, B_1\rangle \leqslant \langle A_2, B_2\rangle \text{ iff } A_1 \subseteq A_2 \text{ (or, equivalently, iff } B_2 \subseteq B_1). \tag{7}$$

If $\langle A_1, B_1\rangle \leqslant \langle A_2, B_2\rangle$ then $\langle A_1, B_1\rangle$ is called a subconcept of $\langle A_2, B_2\rangle$. The set $\mathcal{B}(X, Y, I)$ together with $\leqslant$is called a concept lattice [33]. A concept lattice is a complete lattice whose structure is described by the Basic Theorem of Concept Lattices [10]. The concept lattice is a formalization of a hierarchy of concepts that are found in the input data represented by I. FCA has been applied in many disciplines to analyze object-attribute data including program analysis and software engineering [31,32] and evaluation of questionnaires [6]. Another source of applications of formal concepts comes from data mining. The task of listing all formal concepts is closely related to mining of association rules [1]. Namely, the frequent closed itemsets which appear in mining nonredundant association rules [1,25,34] can be identified with intents of formal concepts whose extents are sufficiently large. Recently, it has been shown in [7] that formal concepts can be used to find optimal factorization of Boolean matrices. In fact, formal concepts correspond with optimal solutions to the discrete basis problem discussed by Miettinen et al. [28]. Finding formal concepts is therefore an important task. The algorithm we propose in this paper behaves well on both sparse and dense incidence data (of reasonable size).

This paper is organized as follows. In Section 2 we recall CbO and introduce the canonicity test. Section 3 describes the new algorithm, shows its correctness, and comments on the relationship to other algorithms. In Section 4 we discuss complexity and efficiency issues, and present an experimental evaluation of the performance of the algorithm.

2. Canonicity test and CbO

In this section we recall CbO [19,21] and the canonicity test. The next section will describe the new algorithm. In the sequel, we assume that $X = \{0, 1, \ldots, m\}$ and $Y = \{0, 1, \ldots, n\}$ are finite nonempty sets of objects and attributes, respectively, and $I \subseteq X \times Y$. Since I is fixed, the concept-forming operators $^{\uparrow_I}$ and $^{\downarrow_I}$ defined by (5) and (6) will be denoted just by $^{\uparrow}$ and $^{\downarrow}$, respectively. The set of all formal concept in I will be denoted by $\mathcal{B}(X, Y, I)$.

CbO has been introduced in [19] (a paper in Russian) and later used and described in [21]. The algorithm is also related to the algorithm proposed by Norris [30] which can be seen as an incremental variant of CbO. CbO lists all formal concepts by a systematic search in the space of all formal concepts, avoiding to list the same concept multiple times by performing a canonicity test. Conceptually, CbO is similar to NextClosure [9] because it uses the same canonicity test but NextClosure lists concepts in a different order. In [21], CbO is described in terms of backtracking. In this section we are going to use a simplified version of CbO introduced in [15] which is formalized by a recursive procedure performing a depth-first search in the space of all formal concepts. This type of description will shed more light on the new algorithm.

The core of CbO is a recursive procedure GenerateFrom, see Algorithm 1. The procedure accepts a formal concept $\langle A,B\rangle$ (an initial formal concept) and an attribute $y \in Y$ (first attribute to be processed) as its arguments. The procedure recursively descends through the space of formal concepts, beginning with $\langle A,B\rangle$.

Algorithm 1: Procedure GenerateFrom($\langle A,B\rangle$,y)

```
1  list ⟨A,B⟩ (e.g., print ⟨A,B⟩ on the screen);
2  if B = Y or y > n then
3    return
4  end
5  for j from y upto n do
6    if j ∉ B then
7      set C to A ∩ {j}↓;
8      set D to C↑;
9      if B ∩ Y_j = D ∩ Y_j then
10       GenerateFrom(⟨C,D⟩, j + 1);
11     end
12   end
13 end
14 return
```

When invoked with $\langle A,B\rangle$ and $y \in Y$, GenerateFrom first lists $\langle A,B\rangle$ (line 1) and then it checks its halting condition (lines 2–4). The computation stops either when $\langle A,B\rangle$ equals $\langle Y^{\downarrow},Y\rangle$ (the least formal concept has been reached) or $y > n$ (there are no more remaining attributes to be processed). Otherwise, the procedure goes through all attributes $j \in Y$ such that $j \geqslant y$ which are not present in the intent B (lines 5 and 6). For each such $j \in Y$, a new formal concept $\langle C,D\rangle = \langle A \cap \{j\}^{\downarrow},(A \cap \{j\}^{\downarrow})^{\uparrow}\rangle$ is computed (lines 7 and 8). After obtaining $\langle C,D\rangle$, the algorithm uses the canonicity test to check whether it should continue with $\langle C,D\rangle$ by recursively calling GenerateFrom or whether $\langle C,D\rangle$ should be "skipped". The canonicity test (line 9) is based on comparing $B \cap Y_j = D \cap Y_j$ where $Y_j \subseteq Y$ is defined by

$$Y_j = \{y \in Y | y < j\}. \tag{8}$$

If the test passes, GenerateFrom is called with $\langle C,D\rangle$ and $j+1$, otherwise, the loop between lines 5–13 continues with the next value of j. The algorithm is correct: if GenerateFrom is invoked with $\langle \emptyset^{\downarrow},\emptyset^{\downarrow\uparrow}\rangle$ and 0, it lists each formal concept exactly once, i.e., the canonicity test prevents a concept from being listed multiple times. The proof for the original CbO is elaborated in [18].

Since we have formulated the algorithm as a recursive procedure rather than using backtracking, we provided an independent proof of its correctness using so-called derivations which we introduced in [15] for the purpose of analysis of parallel implementations of CbO. Recall from [15] that derivations correspond to recursive invocations of GenerateFrom. In a more detail, for $\langle A_1,B_1\rangle$, $\langle A_2,B_2\rangle \in \mathcal{B}(X,Y,I)$ and integers $y_1,y_2 \in Y \cup \{n+1\}$ let $\langle\langle A_1,B_1\rangle,y_1\rangle \vdash \langle\langle A_2,B_2\rangle,y_2\rangle$ denote that for $m = y_2 - 1$ the following conditions

(i) $m \notin B_1$,
(ii) $y_1 < y_2$,
(iii) $B_2 = (B_1 \cup \{m\})^{\downarrow\uparrow}$, and
(iv) $B_1 \cap Y_m = B_2 \cap Y_m$ where Y_m is defined by (8)

are all satisfied. A derivation of $\langle A,B\rangle \in \mathcal{B}(X,Y,I)$ of length $k+1$ is any sequence

$$\langle\langle\emptyset^{\downarrow},\emptyset^{\downarrow\uparrow}\rangle,0\rangle = \langle\langle A_0,B_0\rangle,y_0\rangle,\langle\langle A_1,B_1\rangle,y_1\rangle,\ldots,\langle\langle A_k,B_k\rangle,y_k\rangle = \langle\langle A,B\rangle,y_k\rangle \tag{9}$$

such that $\langle\langle A_i,B_i\rangle,y_i\rangle \vdash \langle\langle A_{i+1},B_{i+1}\rangle,y_{i+1}\rangle$ for each $i = 0,\ldots,k-1$. If $\langle A,B\rangle$ has a derivation of length k we say that $\langle A,B\rangle$ is derivable in k steps.

We can prove the following

Theorem 1 (Existence and Uniqueness of Derivations [15]). *Each $\langle A,B\rangle \in \mathcal{B}(X,Y,I)$ has exactly one derivation. Namely, the derivation of the form (9) in which $y_i = m_i + 1$ and $m_i = \min\{y \in B | y \notin B_{i-1}\}$ hold for all $0 < i \leqslant k$.* □

There is a correspondence between derivations and consecutive invocations of the procedure GenerateFrom. Namely, $\langle\langle A,B\rangle,y\rangle \vdash \langle\langle C,D\rangle,k\rangle$ iff the invocation of GenerateFrom($\langle A,B\rangle$, y) causes GenerateFrom($\langle C,D\rangle$, k) to be called in line 10 of Algorithm 1. Indeed, (i) ensures that the condition in line 6 of Algorithm 1 is satisfied, (ii) corresponds to the fact that the loop between lines 5–13 goes from y upwards, (iii) says that D is the intent computed in line 8 because

$$D = (B \cup \{m\})^{\downarrow\uparrow} = (B \cup \{k-1\})^{\downarrow\uparrow} = (A \cap \{k-1\}^{\downarrow})^{\uparrow} = C^{\uparrow}$$

and (iv) is true iff the condition in line 9 is true.

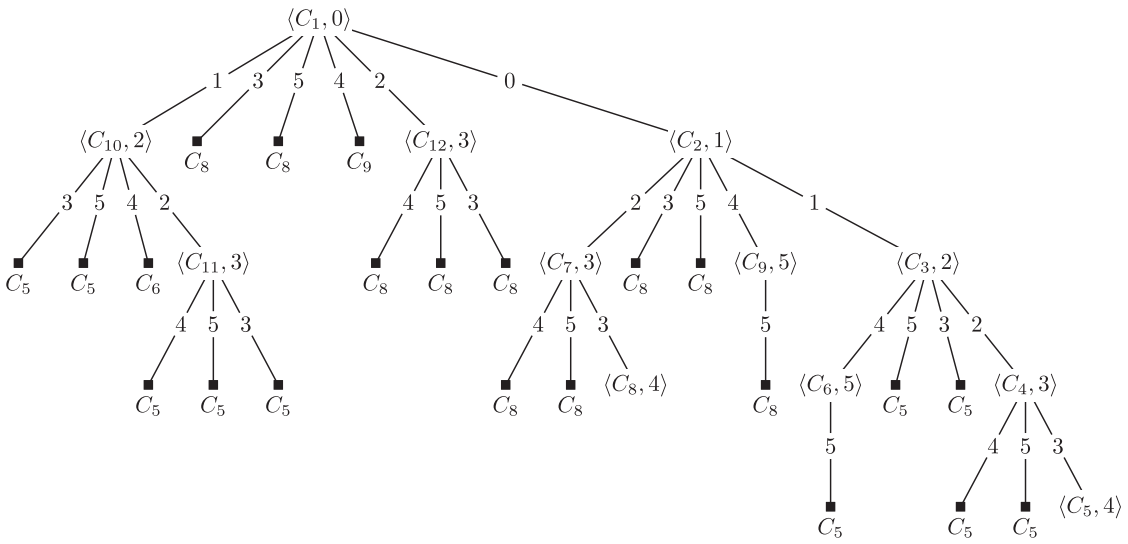

Fig. 1. Call tree of GenerateFrom for $I \subseteq X \times Y$ from Example 1.

The computation of Algorithm 1 and the corresponding derivations can be depicted by a tree as that in Fig. 1. The tree contains two types of nodes: (i) nodes represented by couples $\langle C_i, y_i \rangle$ corresponding to invocations of GenerateFrom with the arguments C_i (a formal concept) and y_i (an attribute), and (ii) leaf nodes denoted by black squares representing computed concepts for which the canonicity test fails. Edges in the tree are labeled by the values of j which are used to compute (new) formal concepts, see lines 7 and 8 of Algorithm 1. That is, nodes $\langle C_i, y_i \rangle$ and $\langle C_j, y_j \rangle$ are connected by an edge with label k iff $\langle C_i, y_i \rangle \vdash \langle C_j, y_j \rangle$ and $y_j = k + 1$. We call such a tree a call tree of GenerateFrom for a given $I \subseteq X \times Y$. It is easily seen that each path from the root of the tree to any node labeled by $\langle C_i, y_i \rangle$ corresponds to a derivation of $\langle C_i, y_i \rangle$. Due to Theorem 1, the nodes labeled by $\langle C_i, y_i \rangle$ are always in a one-to-one correspondence with formal concepts in $\mathcal{B}(X, Y, I)$, showing that the Algorithm 1 is correct. Let us note that there is a correspondence between a call tree like that in Fig. 1 and a CbO-tree described in [21]: our derivations correspond to canonical paths in the CbO-tree. Moreover, paths which are not canonical according to [21] can be seen as paths from the root node of the call tree of GenerateFrom to nodes labeled by black squares.

Example 1. Algorithm 1 and derivations are further demonstrated by the following example. Consider a set $X = \{0, \ldots, 3\}$ of objects and a set $Y = \{0, \ldots, 5\}$ of attributes. An incidence relation $I \subseteq X \times Y$ is given by the following table:

I	0	1	2	3	4	5
0	×	×	×			
1	×		×	×	×	×
2	×	×			×	
3		×	×			

where rows correspond to objects from X, columns correspond to attributes from Y, and table entries "×" or "blank" indicate whether for an object x and an attribute y we have $\langle x, y \rangle \in I$ or $\langle x, y \rangle \notin I$, respectively. The concept-forming operators $^{\uparrow} : 2^X \to 2^Y$ and $^{\downarrow} : 2^Y \to 2^X$ induced by such I have 12 fixpoints:

$$
\begin{aligned}
&C_1 = \langle \{0,1,2,3\}, \emptyset \rangle, && C_5 = \langle \emptyset, \{0,1,2,3,4,5\} \rangle, && C_9 = \langle \{1,2\}, \{0,4\} \rangle,\\
&C_2 = \langle \{0,1,2\}, \{0\} \rangle, && C_6 = \langle \{2\}, \{0,1,4\} \rangle, && C_{10} = \langle \{0,2,3\}, \{1\} \rangle,\\
&C_3 = \langle \{0,2\}, \{0,1\} \rangle, && C_7 = \langle \{0,1\}, \{0,2\} \rangle, && C_{11} = \langle \{0,3\}, \{1,2\} \rangle,\\
&C_4 = \langle \{0\}, \{0,1,2\} \rangle, && C_8 = \langle \{1\}, \{0,2,3,4,5\} \rangle, && C_{12} = \langle \{0,1,3\}, \{2\} \rangle.
\end{aligned}
$$

The concepts are numbered as they are listed by procedure GenerateFrom. Notice that $C_1 = \langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$ represents the initial formal concept which is processed by GenerateFrom. The corresponding call tree can be found in Fig. 1. One can read from the tree that, for example, $\langle C_1, 0 \rangle \vdash \langle C_2, 1 \rangle$, $\langle C_2, 1 \rangle \vdash \langle C_3, 2 \rangle$, and $\langle C_3, 2 \rangle \vdash \langle C_6, 5 \rangle$. Therefore, $\langle C_1, 0 \rangle$, $\langle C_2, 1 \rangle$, $\langle C_3, 2 \rangle$, $\langle C_6, 5 \rangle$ is a derivation and C_6 is derivable in 4 steps. The dataset used in this example will be used to illustrate our improvement of the canonicity test. ■

3. Improved canonicity test and FCbO

In this section, we propose an improvement of the canonicity test used by CbO that reduces the number of formal concepts computed multiple times. In a call tree like that in Fig. 1, such formal concepts are depicted by the black-square nodes.

Our new test and the improved algorithm will reduce the number of such nodes in the call tree without altering the rest of the tree. The major problem with the original canonicity test used by CbO is that it is always used *after* a new formal concept is computed, i.e., after performing the operation of computing a new fixpoint of $^{\downarrow\uparrow}$. We propose to employ an additional test that can be performed *before* a new formal concept is computed, eliminating thus the expensive computation of fixpoints.

3.1. Fast canonicity test

Let us first inspect the canonicity test

$$B \cap Y_j = D \cap Y_j \tag{10}$$

that appears in line 9 of Algorithm 1. Since $^{\downarrow\uparrow}$ is a closure operator and $D = (B \cup \{j\})^{\downarrow\uparrow}$, the monotony of $^{\downarrow\uparrow}$ yields $B \subseteq D$. Thus, it is sufficient to check just the inclusion $B \cap Y_j \supseteq D \cap Y_j$ instead of (10). In other words, the test succeeds iff D and B agree on all attributes which are smaller than j. Hence, the test (10) fails (i.e., the equality is not true) iff the fixpoint $D = (B \cup \{j\})^{\downarrow\uparrow}$ contains an attribute which is "before j" and the attribute is not present in B. Let us denote all such attributes by $B \oslash j$, i.e.

$$B \oslash j = (D \setminus B) \cap Y_j = \left((B \cup \{j\})^{\downarrow\uparrow} \setminus B\right) \cap Y_j. \tag{11}$$

The following lemma shows that knowing that (10) fails for given B and $j \notin B$, we can conclude that the test will also fail for each $B' \supseteq B$ with $j \notin B'$ as long as $B \oslash j$ contains an attribute which is not in B':

Lemma 2 (*On Test Failure Propagation*). *Let $B \subseteq Y$, $j \notin B$, and $B \oslash j \neq \emptyset$. Then, for each $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \not\subseteq B'$, we have $B' \oslash j \neq \emptyset$.*

Proof. Notice that $B \oslash j = (D \setminus B) \cap Y_j \neq \emptyset$ for $D = (B \cup \{j\})^{\downarrow\uparrow}$ means that (10) fails for such B, D and $j \notin B$. Take any $B' \supseteq B$ such that $j \notin B'$ and $B \oslash j \not\subseteq B'$. Let $D' = (B' \cup \{j\})^{\downarrow\uparrow}$. Since $j \notin B'$, we get $B' \subset D'$. In order to show that $B' \oslash j \neq \emptyset$, we prove that $B' \cap Y_j \subset D' \cap Y_j$. Since $B \oslash j \not\subseteq B'$, there is an attribute $y \in B \oslash j$ such that $y \notin B'$. Thus, it suffices to prove that $y \in D' \cap Y_j$. The fact that $y \in Y_j$ follows directly from $y \in B \oslash j = (D \setminus B) \cap Y_j$. Moreover, $y \in B \oslash j$ yields $y \in D$. Using monotony of the closure operator $^{\downarrow\uparrow}$, we get $y \in D = (B \cup \{j\})^{\downarrow\uparrow} \subseteq (B' \cup \{j\})^{\downarrow\uparrow} = D'$, proving the claim. Altogether, $B' \cap Y_j \subset D' \cap Y_j$, i.e. $B' \oslash j \neq \emptyset$. □

Based on Lemma 2, we get the following characterization of derivations:

Theorem 3 (On Nonexistence of Derivations). *Let $\langle\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle, 0\rangle, \ldots, \langle\langle A, B\rangle, y\rangle$ be a derivation and let $j \geqslant y$ be such that $j \notin B$ and $B \oslash j \neq \emptyset$. Then there is no derivation*

$$\langle\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle, 0\rangle, \ldots, \langle\langle A, B\rangle, y\rangle, \ldots, \langle\langle A', B'\rangle, y'\rangle, \langle\langle C', D'\rangle, j+1\rangle,$$

where $B \oslash j \not\subseteq B'$.

Proof. The claim is a consequence of Lemma 2. Indeed, take arbitrary $B' \supseteq B$ such that $B \oslash j \not\subseteq B'$. Assume there is a sequence

$$\langle\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle, 0\rangle, \ldots, \langle\langle A, B\rangle, y\rangle, \ldots, \langle\langle A', B'\rangle, y'\rangle$$

which is a derivation of $\langle A', B'\rangle$. We can prove that the derivation cannot be extended by $\langle\langle C', D'\rangle, j+1\rangle$. By contradiction, assume that $\langle\langle A', B'\rangle, y'\rangle \vdash \langle\langle C', D'\rangle, j+1\rangle$. By definition of "$\vdash$", we get $D' = (B' \cup \{j\})^{\downarrow\uparrow}$ and $B' \cap Y_j = D' \cap Y_j$, i.e., $B' \oslash j = (D' \setminus B') \cap Y_j = \emptyset$. On the other hand, we have assumed $B \oslash j \not\subseteq B'$, i.e. Lemma 2 yields $B' \oslash j \neq \emptyset$, a contradiction to $B' \oslash j = \emptyset$. □

The result shown in Theorem 3 allows us to split the canonicity test into two parts: First part which is quick and does not require computing closures and a second part which is basically the original canonicity test. Indeed, according to Theorem 3, if we know that $B \oslash j \neq \emptyset$ for some $j \notin B$ then having a derivation

$$\langle\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle, 0\rangle, \ldots, \langle\langle A, B\rangle, y\rangle, \ldots, \langle\langle A', B'\rangle, y'\rangle$$

with $B \oslash j \not\subseteq B'$, we automatically know (without computing any closures) that it cannot be further extended by $\langle\langle C', D'\rangle, j+1\rangle$. In other words, $D' = (B' \cup \{j\})^{\downarrow\uparrow}$ is not computed at all. Therefore, the first part of the new test uses the observation of Theorem 3. If the first part of the test cannot be applied because $B \oslash j = \emptyset$, we still have to perform the second part of the test, i.e., the original canonicity test which involves computing the closure $(B' \cup \{j\})^{\downarrow\uparrow}$. Nevertheless, we will see in Section 4 that the number of cases in which we actually perform the original canonicity test is surprisingly low compared to the number of quick tests based on Theorem 3. The idea of the new combined canonicity test is further illustrated by the following example.

Example 2. Consider the input data from Example 1 and the corresponding call tree in Fig. 1. If we apply the new canonicity test based on Theorem 3, we in fact perform a particular tree pruning in which we omit some of the black-square leaf nodes of the tree. The result is shown in Fig. 2. The bold edges are those which remain in the call tree. The leaf nodes that are omitted are denoted in gray and the corresponding edges are dotted. Notice that not all black-square leaf nodes are omitted.

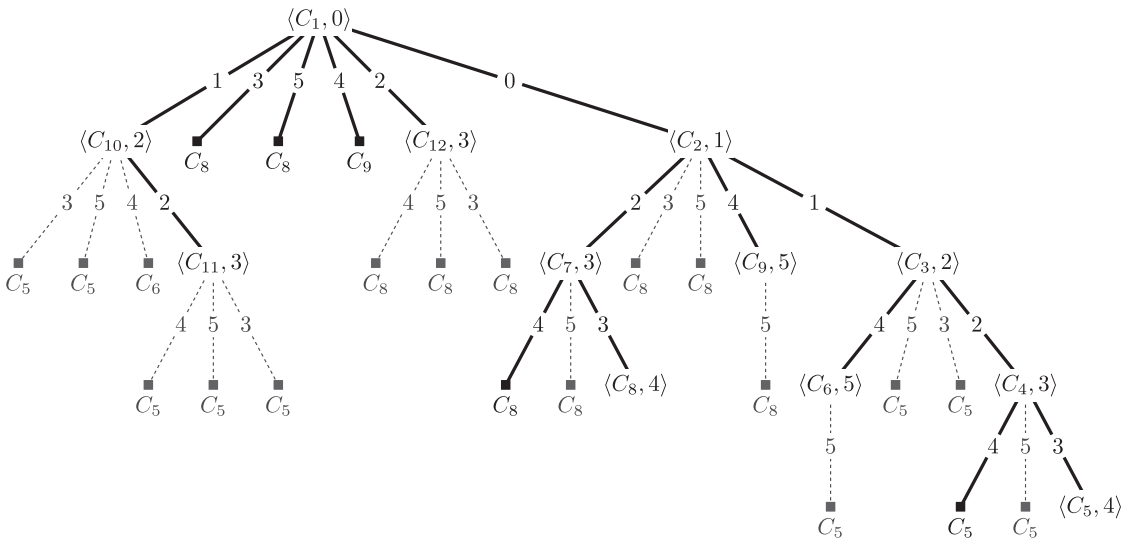

Fig. 2. Example of a call tree with a reduced number of leaf nodes.

C_8 appears three times as a leaf node, C_9 and C_5 appear once, meaning that the formal concept C_8 is computed four times during the computation and both C_9 and C_5 are computed twice. The total number of closures computed during the computation is 17 which is a significant reduction compared to the 34 nodes of the original call tree in Fig. 1.

Let us outline how the new test is used to prune the tree. Consider the first formal concept $C_1 = \langle A_1, B_1\rangle = \langle\{0,1,2,3\},\emptyset\rangle$, see Example 1 for the list of all concepts. At this point, we perform the usual canonicity test because we have no information from previous levels of the tree (we are on the top of the tree). For $j \in \{0,1,2\}$, the test succeeds. For instance, in case of $j = 2$, we get $C_{12} = \langle A_{12}, B_{12}\rangle = \langle\{0,1,3\},\{2\}\rangle$, i.e. $B_1 \cap Y_2 = \emptyset = B_{12} \cap Y_2$. On the other hand, the test fails for $j \in \{3,4,5\}$. For instance, in case of $j = 3$, we get $C_8 = \langle A_8, B_8\rangle = \langle\{1\},\{0,2,3,4,5\}\rangle$ and hence $B_1 \cap Y_3 = \emptyset \neq \{0,2\} = B_8 \cap Y_3$. Therefore, $B_1 \otimes 3 = \{0,2\}$. Analogously, we get $B_1 \otimes 4 = \{0\}$ and $B_1 \otimes 5 = \{0,2,3,4\}$. The sets $B_1 \otimes 3$, $B_1 \otimes 4 = \{0\}$, and $B_1 \otimes 5$ can be further used to prune the tree according to Theorem 3. Indeed, consider the tree node $\langle C_{10}, 2\rangle$. Since $C_{10} = \langle A_{10}, B_{10}\rangle = \langle\{0,2,3\},\{1\}\rangle$, we get $B_1 \otimes 3 \not\subseteq B_{10}$, $B_1 \otimes 4 \not\subseteq B_{10}$, and $B_1 \otimes 5 \not\subseteq B_{10}$, i.e. neither $j \in \{3,4,5\}$ can be used to extend the derivation. In case of $j = 2$, we perform the usual canonicity test which is successful. In a similar way, the tree can be pruned beginning with the other nodes $\langle C_i, y_i\rangle$.

The fast test based on Theorem 3 is not always applicable. It is evident that we cannot apply the test on the top-most level of the call tree. There are, however, situations where it cannot be applied on deeper levels as well. Consider, e.g., the tree node $\langle C_7, 3\rangle$ where $C_7 = \langle A_7, B_7\rangle = \langle\{0,1\},\{0,2\}\rangle$. Since $B_1 \otimes 4 = \{0\} \subseteq B_7$, Theorem 3 cannot be applied. On the other hand, if we perform the original canonicity test with B_7 and $(B_7 \cup \{4\})^{\downarrow\uparrow} = \{0,2,3,4,5\} = B_8$, we get $B_7 \cap Y_4 = \{0,2\} \neq \{0,2,3\} = B_8 \cap Y_4$, i.e., the derivation cannot be extended by $\langle C_8, 5\rangle$ but in order to see this we had to compute the closure $(B_7 \cup \{4\})^{\downarrow\uparrow} = B_8$ which should be considered an expensive operation (especially in case of large data sets). A similar situation appears in case of the node $\langle C_4, 3\rangle$ and $j = 4$, cf. Fig. 2. ■

3.2. Modified algorithm

In this section, we describe how the new canonicity test based on Theorem 3 can be implemented in an extended version of CbO. As Example 2 shows, during the computation we have to propagate the information about sets $B_i \otimes y_i$ which take part in the new test. In particular, the information must be propagated in the top-down direction, from the root node of the call tree to the leaves. As a consequence, we have to change the search strategy of the algorithm (the depth-first search in the space of concepts as it is used in CbO is no longer useful), resulting in a new algorithm called FCbO ("F" stands for "Fast").

Remark 1. A call tree is a diagram depicting recursive calls of GenerateFrom. Consecutive invocations of GenerateFrom correspond to the depth-first search in the call tree. For instance, in case of node $\langle C_1, 0\rangle$ in Fig. 1, GenerateFrom continues with the subtree with root node $\langle C_2, 1\rangle$. After the whole subtree is processed, it continues with the subtree with root node $\langle C_{10}, 2\rangle$, etc. The problem with this behavior is that in order to apply the new canonicity test in the subtree with root $\langle C_2, 1\rangle$, we shall already have the information about $B_1 \otimes 3 = \{0,2\}$. Analogously, we get $B_1 \otimes 4 = \{0\}$ and $B_1 \otimes 5 = \{0,2,3,4\}$, see Example 2, which is available *only after* we process all attributes in the invocation of $\langle C_1, 0\rangle$. Therefore, we are going to modify GenerateFrom so that instead of the recursive calls, it stores information about computed concepts in a queue. Then, after all attributes are processed, it performs a recursive invocation for each concept in the queue. This effectively changes the order in which we compute new concepts because we use a combined depth-first and breadth-first search in the call tree but it *does not* change the order of listing of formal concepts because the listing appears *after* each recursive call, as in CbO. ■

Algorithm 2: Procedure FastGenerateFrom($\langle A, B\rangle$, y, $\{N_y | y \in Y\}$)

```
1  list ⟨A,B⟩ (e.g., print A and B on screen);
2  if B = Y or y > n then
3    return
4  end
5  for j from y upto n do
6    set M_j to N_j;
7    if j ∉ B and N_j ∩ Y_j ⊆ B ∩ Y_j then
8      set C to A ∩ {j}^↓;
9      set D to C^↑;
10     if B ∩ Y_j = D ∩ Y_j then
11       put ⟨⟨C,D⟩,j + 1⟩ to queue;
12     else
13       set M_j to D;
14     end
15   end
16 end
17 while get ⟨⟨C,D⟩,j⟩ from queue do
18   FastGenerateFrom(⟨C, D⟩, j, {M_y|y ∈ Y});
19 end
20 return
```

We are going to represent FCbO by a recursive procedure FastGenerateFrom, see Algorithm 2. The procedure accepts three arguments: a formal concept $\langle A,B\rangle$ (an initial formal concept), an attribute $y \in Y$ (first attribute to be processed), and a set $\{N_y \subseteq Y | y \in Y\}$ of subsets of attributes Y. The intended meaning of the first two arguments is the same as in case of GenerateFrom, see Algorithm 1. The purpose of the third argument is to carry information about attributes in sets $B_i \oslash y_i$. The precise meaning of N_y will be specified later. Each invocation of FastGenerateFrom uses the following *local variables*: a *queue* as a temporary storage for computed concepts and sets of attributes M_y ($y \in Y$) which are used in place of the third argument for further invocations of FastGenerateFrom.

When invoked with $\langle A,B\rangle$, $y \in Y$, and $\{N_y | y \in Y\}$, FastGenerateFrom first processes $\langle A,B\rangle$ and then it checks the same halting condition as GenerateFrom, see lines 1–4. If the computation does not halt, the procedure goes through all attributes $j \in Y$ such that $j \geqslant y$, see lines 5–16. For each such j, the procedure creates a local copy M_j of the set N_j (line 6). If $j \notin B$, a test based on Theorem 3 is performed by checking $N_j \cap Y_j \subseteq B \cap Y_j$. If the test succeeds, the procedure goes on with computing a new formal concept $\langle C,D\rangle$, see lines 8 and 9. Then it performs the original canonicity test (line 10). If the test is positive, the formal concept $\langle C,D\rangle$ together with the attribute $j + 1$ are stored in a *queue* (line 11). Otherwise, M_j is set to D (line 13). Notice that the loop between lines 5–15 does not perform any recursive calls of FastGenerateFrom. Instead, the information about computed concepts and attributes used to generate the concepts is stored in the *queue*. The recursive invocations of FastGenerateFrom are performed after all the attributes are processed. Indeed, the loop between lines 17–19 goes over all records in the *queue* and recursively calls FastGenerateFrom with arguments being the new concept, new starting attribute, and new set of subsets $\{M_y | y \in Y\}$ of attributes.

In order to list all formal concepts, we invoke Algorithm 2 with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$, $y = 0$ and $\{N_y = \emptyset | y \in Y\}$ as its initial arguments. The following assertion says that the algorithm is correct:

Theorem 4 (Correctness of FCbO). *When invoked with $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$, $y = 0$, and $\{N_y = \emptyset | y \in Y\}$, Algorithm 2 lists all formal concepts in $\langle X, Y, I\rangle$, each of them exactly once.*

Proof. Since Algorithm 1 (CbO) is correct, is it sufficient to show that Algorithm 2 (FCbO) does not omit any formal concept during the computation. Thus, we have to check that the new canonicity test is applied correctly. The rest follows from the correctness of Algorithm 1, in particular the existence and uniqueness of derivations, see Theorem 1. Let us inspect the values of N_j's and M_j's during each invocation of FastGenerateFrom. During the first invocation, $\{N_y = \emptyset | y \in Y\}$, i.e. $N_j \cap Y_j = \emptyset \subseteq B \cap Y_j$ is trivially true, i.e. each attribute $j \notin B$ is processed between lines 8–14. As one can see, during each invocation of FastGenerateFrom, the value of M_j is either equal to N_j (we say that the value of M_j is *inherited* from previous invocation) or M_j equals $D = (B \cup \{j\})^{\downarrow\uparrow}$ (we say that the value of M_j is *updated* in the current invocation). If M_j is updated then M_j is the intent of a formal concept $\langle C,D\rangle$ which fails the canonicity test in line 10. Therefore, it is easy to see that during an invocation of FastGenerateFrom($\langle A, B\rangle, y, \{N_y | y \in Y\}$), for each $j \geqslant y$, either $N_j = \emptyset$ or there is a formal concept $\langle A^*, B^*\rangle$ such that the following hold

(i) $B^* \subseteq B$,
(ii) $B^* \oslash j \neq \emptyset$, and
(iii) $N_j = (B^* \cup \{j\})^{\downarrow\uparrow}$.

Notice that from (iii) it follows that $B^* \oslash j = ((B^* \cup \{j\})^{\downarrow\uparrow} \setminus B^*) \cap Y_j = (N_j \setminus B^*) \cap Y_j$. Hence, in order to prove correctness, it suffices to show that the condition $N_j \cap Y_j \subseteq B \cap Y_j$ present in line 7 of Algorithm 2 fails iff $B^* \oslash j \not\subseteq B$ which appears in Theorem 3 as a necessary condition for pruning. Therefore, we prove the following

Claim 1.
$B^* \oslash j \subseteq B$ iff $N_j \cap Y_j \subseteq B \cap Y_j$:

"⇒": Let $B^* \oslash j \subseteq B$. Using (iii), we get $(N_j \setminus B^*) \cap Y_j \subseteq B$. Furthermore, (i) yields $N_j \setminus B \subseteq N_j \setminus B^*$, i.e. we obtain $(N_j \setminus B) \cap Y_j \subseteq B$. The last inclusion implies that $N_j \cap Y_j \subseteq B \cap Y_j$. Indeed, by contradiction, from $y \in N_j \cap Y_j$ and $y \notin B$ it follows that $y \in N_j$, i.e., $y \in N_j \setminus B$ and thus $y \in (N_j \setminus B) \cap Y_j \subseteq B$ because $y \in Y_j$, contradicting the fact that $y \notin B$. Therefore, we have $N_j \cap Y_j \subseteq B \cap Y_j$. "⇐": Suppose that $B^* \oslash j \not\subseteq B$. Then, there is $y \in B^* \oslash j$ such that $y \notin B$. From $y \in B^* \oslash j$ and (iii), we get $y \in N_j$ and $y \in Y_j$. Therefore, $y \in N_j \cap Y_j$ and $y \notin B \cap N_j$ because $y \notin B$, showing $N_j \cap Y_j \not\subseteq B \cap Y_j$.

Therefore, as a consequence of Theorem 3, if $N_j \cap Y_j \subseteq B \cap Y_j$ fails then we can skip the attribute j because B and $D = (B \cup \{j\})^{\downarrow\uparrow}$ would fail the canonicity test in line 10. Altogether, FCbO lists all formal concepts, each of them exactly once. □

Remark 2. In Algorithm 2, the additional information about attributes that is needed to perform the test is stored in procedure arguments N_y which are, in fact, particular intents. On the other hand, the test formulated in Theorem 3 is based on sets of the form $B^* \oslash j$. We use N_y's instead of sets $B^* \oslash j$ because of efficiency reasons: Since N_y represents an intent of a concept that has already been computed, the third argument $\{N_y = \emptyset | y \in Y\}$ for FastGenerateFrom can be organized as a list (or an array) of references (pointers) to such intents. Storing referenced objects in a linear data structure is much cheaper an operation than computing $B^* \oslash j$ and storing the resulting value. More efficiency issues will be discussed in Section 4. ■

The following example illustrates recursive invocations of FastGenerateFrom during the computation.

Example 3. We demonstrate the computation of Algorithm 2 for the input data from Example 1 by listing important steps of the algorithm. We focus on steps performed in lines 1 (listing of found formal concepts), 7 (quick canonicity test), 11 (putting a new concept to a queue), 13 (updating information about attributes in sets $B_i \oslash y_i$), and 18 (recursive invocations of FastGenerateFrom). In addition to that, if an invocation of FastGenerateFrom terminates either in line 3 or 20, we denote this fact by "⊥" in a separate line. Nested invocations are separated by horizontal indent. In the example, each formal concept is denoted by $C_i = \langle A_i, B_i \rangle$, i.e., each B_i is the intent of the corresponding C_i. The rest of the notation is the same as in Algorithm 2. When FastGenerateFrom is invoked with C_1, 0, and $\{N_y = \emptyset | y \in Y\}$, the computation proceeds as follows:

line 1: **list** C_1 = $\langle \{0,1,2,3\}, \emptyset \rangle$
line 7: trivial success for $j = 0$ because $N_0 = \emptyset$
line 11: put $\langle C_2, 1 \rangle = \langle \langle \{0,1,2\}, \{0\} \rangle, 1 \rangle$ to *queue*
line 7: trivial success for $j = 1$ because $N_1 = \emptyset$
line 11: put $\langle C_{10}, 2 \rangle = \langle \langle \{0,2,3\}, \{1\} \rangle, 2 \rangle$ to *queue*
line 7: trivial success for $j = 2$ because $N_2 = \emptyset$
line 11: put $\langle C_{12}, 3 \rangle = \langle \langle \{0,1,3\}, \{2\} \rangle, 3 \rangle$ to *queue*
line 7: trivial success for $j = 3$ because $N_3 = \emptyset$
line 13: set M_3 to $D = (\emptyset \cup \{3\})^{\downarrow\uparrow} = \{0,2,3,4,5\} = B_8$
line 7: trivial success for $j = 4$ because $N_4 = \emptyset$
line 13: set M_4 to $D = (\emptyset \cup \{4\})^{\downarrow\uparrow} = \{0,4\} = B_9$
line 7: trivial success for $j = 5$ because $N_5 = \emptyset$
line 13: set M_5 to $D = (\emptyset \cup \{5\})^{\downarrow\uparrow} = \{0,2,3,4,5\} = B_8$
line 18: get $\langle C_2, 1 \rangle$ from *queue* and call FASTGENERATEFROM(C_2, 1, $\{M_y | y \in Y\}$)
  line 1: **list** C_2 = $\langle \{0,1,2\}, \{0\} \rangle$
  line 7: trivial success for $j = 1$ because $N_1 = \emptyset$
  line 11: put $\langle C_3, 2 \rangle = \langle \langle \{0,2\}, \{0,1\} \rangle, 2 \rangle$ to *queue*
  line 7: trivial success for $j = 2$ because $N_2 = \emptyset$
  line 11: put $\langle C_7, 3 \rangle = \langle \langle \{0,1\}, \{0,2\} \rangle, 3 \rangle$ to *queue*
  line 7: failure for $j = 3$, $B = \{0\}$, and $N_3 = \{0,2,3,4,5\} = B_8$ because $\{2\} \not\subseteq B$
  line 7: success for $j = 4$, $B = \{0\}$, and $N_4 = \{0,4\} = B_9$
  line 11: put $\langle C_9, 5 \rangle = \langle \langle \{1,2\}, \{0,4\} \rangle, 5 \rangle$ to *queue*
  line 7: failure for $j = 5$, $B = \{0\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{2,3,4\} \not\subseteq B$
  line 18: get $\langle C_3, 2 \rangle$ from *queue* and call FastGenerateFrom(C_3, 2, $\{M_y | y \in Y\}$)
    line 1: **list** C_3 = $\langle \{0,2\}, \{0,1\} \rangle$
    line 7: trivial success for $j = 2$ because $N_2 = \emptyset$
    line 11: put $\langle C_4, 3 \rangle = \langle \langle \{0\}, \{0,1,2\} \rangle, 3 \rangle$ to *queue*

(*continued on next page*)

line 7: failure for $j = 3$, $B = \{0,1\}$, and $N_3 = \{0,2,3,4,5\} = B_8$ because $\{2\} \not\subseteq B$
line 7: success for $j = 4$, $B = \{0,1\}$, and $N_4 = \{0,4\} = B_9$
line 11: put $\langle C_6, 5\rangle = \langle\langle\{2\},\{0,1,4\}\rangle, 5\rangle$ to *queue*
line 7: failure for $j = 5$, $B = \{0,1\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{2,3,4\} \not\subseteq B$
line 18: get $\langle C_4, 3\rangle$ from *queue* and call FastGenerateFrom$(C_4, 3, \{M_y | y \in Y\})$
line 1: **list** C_4 **=** $\langle\{0\},\{0,1,2\}\rangle$
line 7: success for $j = 3$, $B = \{0,1,2\}$, and $N_3 = \{0,2,3,4,5\} = B_8$
line 11: put $\langle C_5, 4\rangle = \langle\ \langle\emptyset,\{0,1,2,3,4,5\}\rangle, 4\rangle$ to *queue*
line 7: success for $j = 4$, $B = \{0,1,2\}$, and $N_4 = \{0,4\} = B_9$
line 13: set M_4 to $D = (\{0,1,2\} \cup \{4\})^{\downarrow\uparrow} = \{0,1,2,3,4,5\} = B_5$
line 7: failure for $j = 5$, $B = \{0,1,2\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{3,4\} \not\subseteq B$
line 18: get $\langle C_5, 4\rangle$ from *queue* and call FastGenerateFrom$(C_5, 4, \{M_y | y \in Y\})$
line 1: **list** C_5 **=** $\langle\emptyset,\{0,1,2,3,4,5\}\rangle$
$\perp$ return from invocation for C_5
$\perp$ return from invocation for C_4
line 18: get $\langle C_6, 5\rangle$ from *queue* and call FastGenerateFrom$(C_6, 5, \{M_y | y \in Y\})$
line 1: **list** C_6 **=** $\langle\{2\},\{0,1,4\}\rangle$
line 7: failure for $j = 5$, $B = \{0,1,4\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{2,3\} \not\subseteq B$
$\perp$ return from invocation for C_6
$\perp$ return from invocation for C_3
line 18: get $\langle C_7, 3\rangle$ from *queue* and call FastGenerateFrom$(C_7, 3, \{M_y | y \in Y\})$
line 1: **list** C_7 **=** $\langle\{0,1\},\{0,2\}\rangle$
line 7: success for $j = 3$, $B = \{0,2\}$, and $N_3 = \{0,2,3,4,5\} = B_8$
line 11: put $\langle C_8, 4\rangle = \langle\langle\{1\},\{0,2,3,4,5\}\rangle, 4\rangle$ to *queue*
line 7: success for $j = 4$, $B = \{0,2\}$, and $N_4 = \{0,4\} = B_9$
line 13: set M_4 to $D = (\{0,2\} \cup \{4\})^{\downarrow\uparrow} = \{0,2,3,4,5\} = B_8$
line 7: failure for $j = 5$, $B = \{0,2\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{3,4\} \not\subseteq B$
line 18: get $\langle C_8, 4\rangle$ from *queue* and call FastGenerateFrom$(C_8, 4, \{M_y | y \in Y\})$
line 1: **list** C_8 **=** $\langle\{1\},\{0,2,\ 3,4,5\}\rangle$
$\perp$ return from invocation for C_8
$\perp$ return from invocation for C_7
line 18: get $\langle C_9, 5\rangle$ from *queue* and call FastGenerateFrom$(C_9, 5, \{M_y | y \in Y\})$
line 1: **list** C_9 **=** $\langle\{1,2\},\{0,4\}\rangle$
line 7: failure for $j = 5$, $B = \{0,4\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{2,3\} \not\subseteq B$
$\perp$ return from invocation for C_9
$\perp$ return from invocation for C_2
line 18: get $\langle C_{10}, 2\rangle$ from *queue* and call FastGenerateFrom$(C_{10}, 2, \{M_y | y \in Y\})$
line 1: **list** C_{10} **=** $\langle\{0,2,3\},\{1\}\rangle$
line 7: trivial success for $j = 2$ because $N_2 = \emptyset$
line 11: put $\langle C_{11}, 3\rangle = \langle\langle\{0,3\},\{1,2\}\rangle, 3\rangle$ to *queue*
line 7: failure for $j = 3$, $B = \{1\}$, and $N_3 = \{0,2,3,4,5\} = B_8$ because $\{0,2\} \not\subseteq B$
line 7: failure for $j = 4$, $B = \{1\}$, and $N_4 = \{0,4\} = B_9$ because $\{0\} \not\subseteq B$
line 7: failure for $j = 5$, $B = \{1\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{0,2,3,4\} \not\subseteq B$
line 18: get $\langle C_{11}, 3\rangle$ from *queue* and call FastGenerateFrom$(C_{11}, 3, \{M_y | y \in Y\})$
line 1: **list** C_{11} **=** $\langle\{0,3\},\{1,2\}\rangle$
line 7: failure for $j = 3$, $B = \{1,2\}$, and $N_3 = \{0,2,3,4,5\} = B_8$ because $\{0\} \not\subseteq B$
line 7: failure for $j = 4$, $B = \{1,2\}$, and $N_4 = \{0,4\} = B_9$ because $\{0\} \not\subseteq B$
line 7: failure for $j = 5$, $B = \{1,2\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{0,3,4\} \not\subseteq B$
$\perp$ return from invocation for C_{11}
$\perp$ return from invocation for C_{10}
line 18: get $\langle C_{12}, 3\rangle$ from *queue* and call FastGenerateFrom$(C_{12}, 3, \{M_y | y \in Y\})$
line 1: **list** C_{12} **=** $\langle\{0,1,3\},\{2\}\rangle$
line 7: failure for $j = 3$, $B = \{2\}$, and $N_3 = \{0,2,3,4,5\} = B_8$ because $\{0\} \not\subseteq B$
line 7: failure for $j = 4$, $B = \{2\}$, and $N_4 = \{0,4\} = B_9$ because $\{0\} \not\subseteq B$
line 7: failure for $j = 5$, $B = \{2\}$, and $N_5 = \{0,2,3,4,5\} = B_8$ because $\{0,3,4\} \not\subseteq B$
$\perp$ return from invocation for C_{12}
$\perp$ return from invocation for C_1

Notice that line 7 is either success or failure depending on the outcome of the new canonicity test. Each occurrence of line 7 is followed either by line 11 or line 13 depending on the outcome of the original canonicity test in line 10 (for brevity, line 10 is not displayed). ■

3.3. On the relationship to NextClosure

Notice that FCbO lists all formal concepts in the same order as CbO. Although FCbO first computes closures which are put in a queue and then makes the appropriate recursive calls, it lists the concepts in the same order as CbO because the listing is performed as a first action after the invocation of FastGenerateFrom. Hence, the listing does not necessarily follow the computation of a closure as it can be seen from Example 3.

The order in which concepts are listed by FCbO can be changed in various ways. For instance, if we move line 1 of Algorithm 2 between lines 11 and 12, the concept will be listed in an order which agrees with the combined breadth-first and depth-first search order of the call tree, see Remark 1.

More importantly, the algorithm can be easily modified to produce formal concepts in the same order as Ganter's NextClosure algorithm [9]. Recall that NextClosure lists all concepts in a lectic order: $B_1 \subseteq Y$ is *lectically smaller* [10] than $B_2 \subseteq Y$, denoted $B_1 <_\ell B_2$, if the smallest element that distinguishes B_1 and B_2 belongs to B_2. That is,

$$B_1 <_\ell B_2 \quad \text{iff} \quad \text{there is } j \in B_2 \setminus B_1 \text{ such that } B_1 \cap Y_j = B_2 \cap Y_j, \tag{12}$$

where Y_j is defined as in (8). It can be shown that $<_\ell$ is a total strict order on 2^Y. NextClosure lists the formal concepts in the (unique) order $<_\ell$ by an iterative computation of lectic successors, starting with the lectically smallest concept $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$. The following claim characterizes nodes of a call tree in terms of their lectic relationship.

Theorem 5. *Let $\{\langle\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle, 0\rangle, \ldots, \langle C, y\rangle, \langle C_i, y_i + 1\rangle | i \in J\}$ be a J-indexed set of derivations of formal concepts C_i with intents B_i. Let B be the intent of C. Then the following are true:*

(i) for each $i \in J$: $B <_\ell B_i$,
(ii) for each $j, k \in J$ with $y_j < y_k$ and each $\langle C^*, y^* + 1\rangle$ such that $\langle C_k, y_k + 1\rangle \vdash^* \langle C^*, y^* + 1\rangle : B^* <_\ell B_j$ where B^* is intent of C^* and $\vdash^*$ is the reflexive and transitive closure of $\vdash$.

Proof. See Fig. 3 for a symbolic schema for the proof.

(i) is easy to see: Since $\langle C, y\rangle \vdash \langle C_i, y_i + 1\rangle$, we have $B \cap Y_{y_i} = B_i \cap Y_{y_i}$. In addition to that, $y_i \in B_i$ and $y_i \notin B$, i.e. (12) is satisfied for j being y_i, showing $B <_\ell B_i$.

(ii) We check that for y_j we have $B^* \cap Y_{y_j} = B_j \cap Y_{y_j}$, $y_j \notin B^*$, and $y_j \in B_j$. From this, we get $B^* <_\ell B_j$ directly from (12). Notice that $y_j < y_k$ implies $Y_{y_j} \subset Y_{y_k}$. Thus, from $B \cap Y_{y_k} = B_k \cap Y_{y_k}$ it follows that $B \cap Y_{y_j} = B_k \cap Y_{y_j}$. Using $B \cap Y_{y_j} = B_j \cap Y_{y_j}$ and the latter equality, we get $B_k \cap Y_{y_j} = B_j \cap Y_{y_j}$. Moreover, from $\langle C_k, y_k + 1\rangle \vdash^* \langle C^*, y^* + 1\rangle$ it follows that $B_k \cap Y_{y_k} = B^* \cap Y_{y_k}$, i.e., $B_k \cap Y_{y_j} = B^* \cap Y_{y_j}$ because $y_j < y_k$. Putting $B_k \cap Y_{y_j} = B^* \cap Y_{y_j}$ and $B_k \cap Y_{y_j} = B_j \cap Y_{y_j}$ together, we get $B^* \cap Y_{y_j} = B_j \cap Y_{y_j}$. Thus, it remains to show that $y_j \in B_j$ and $y_j \notin B^*$. The first claim is evident. In order to prove $y_j \notin B^*$, observe that $y_j \notin B$ and consequently $y_j \notin B \cap Y_{y_k} = B_k \cap Y_{y_k} = B^* \cap Y_{y_k}$, meaning that $y_j \notin B^*$ because $y_j < y_k$. □

Theorem 5 shows how the call tree should be traversed if anyone wants to list all concepts according to $<_\ell$. If we take the concepts C and $\{C_i | i \in J\}$ as in Theorem 5, we can see that C should be listed before all C_i's due to (i). Furthermore, if we take two different concepts C_j and C_k ($j, k \in J$), C_k should be listed before C_j iff $y_j < y_k$ because of (ii). Therefore, (i) and (ii) mean that given a subtree of a call tree, the root node must be listed first and the descending nodes C_i should be listed in a descending order according to y_i. Furthermore, (ii) says that each node C^* derivable from C_k should also be listed before C_j and, at the same time, after C_k because of (i). This means that in order to list all formal concepts in a lectic order, we have to perform a depth-first search through the call tree, assuming that we process all attributes in the descending order. Since Algorithm 2 already performs the depth-first search, it suffices to ensure the descending order of processed attributes. We can do that by modifying Algorithm 2 in one of the following ways:

(i) we can use a *stack* instead of a *queue* to store computed formal concepts, or
(ii) we can modify the loop in line 5 so that it goes "**from** n **downto** y".

This way for obtaining concepts in a lectic order is much faster than the iterative algorithm from [10] because we can compute fixpoints of $^{\downarrow\uparrow}$ more efficiently, see Section 4, and we employ the fast canonicity test. Performance comparisons can be found in Section 4.2.

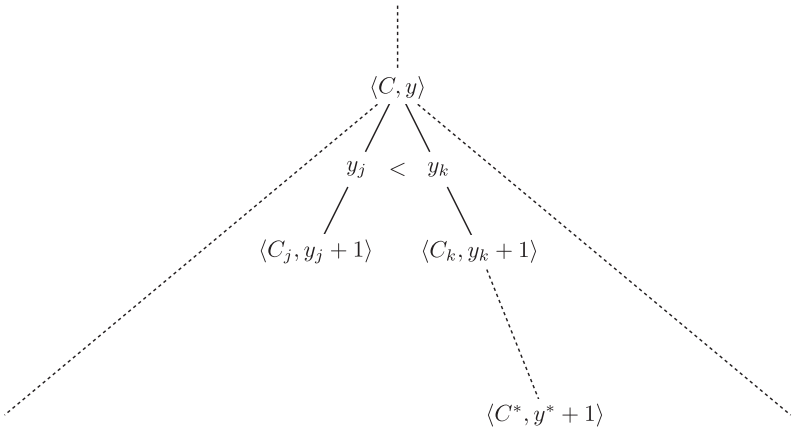

Fig. 3. Schema for the proof of Theorem 5.

3.4. On the relationship to AddIntent

In [27], the authors have introduced an incremental algorithm AddIntent for computing formal concepts together with the subconcept–superconcept hierarchy $\leqslant$ given by (7). Unlike FCbO, the algorithm proposed in [27] is incremental, i.e., it incrementally computes the concept lattice of given context by adding all attributes (or objects as in [27]) one by one. Interestingly, AddIntent includes a particular optimization that is analogous to the fast canonicity test of FCbO introduced in this paper. The key difference is that AddIntent uses a slightly different canonicity test that is based on the ordering $\leqslant$ of formal concepts (7), whereas FCbO uses the order of processed attributes. The approach used by AddIntent is more beneficial if one wants to compute the whole concept lattice instead of just computing the formal concepts. On the other hand, the approach taken by FCbO is simpler and is more efficient if only the set of formal concepts is considered.

Using the notation from our paper, [27] defines a notion of a canonical generator of a formal concept $\langle C,D\rangle$ which can be described as follows. First, denote by $\mathcal{B}_i(X,Y_i,I_i)$ the set of all formal concepts of a formal context $\langle X,Y_i,I_i\rangle$ where $I_i = I \cap (X \times Y_i)$ with Y_i defined as in (8) (i.e., $\langle X,Y_i,I_i\rangle$ represents the original context restricted to attributes $0,\ldots,i-1$). Then, $\langle C,D\rangle \in \mathcal{B}_{i+1}(X,Y_{i+1},I_{i+1})$ is called new in $\mathcal{B}_{i+1}(X,Y_{i+1},I_{i+1})$ if C is distinct from all concept extents from $\mathcal{B}_i(X,Y_i,I_i)$. Furthermore, if $\langle C,D\rangle$ is new in $\mathcal{B}_{i+1}(X,Y_{i+1},I_{i+1})$, then $\langle A,B\rangle \in \mathcal{B}_i(X,Y_i,I_i)$ is called a generator of $\langle C,D\rangle$ if $D = (B \cup \{i\})^{\downarrow_{i+1}\uparrow_{i+1}}$ and thus $C = A \cap \{i\}^{\downarrow_{i+1}}$. A canonical generator $\langle A,B\rangle$ of $\langle C,D\rangle$ is then the infimum

$$\langle A,B\rangle = \left\langle \bigcap_{j\in J} A_j, \left(\bigcup_{j\in J} B_j\right)^{\downarrow_I\uparrow_I} \right\rangle$$

of all generators $\langle A_j,B_j\rangle \in \mathcal{B}_i(X,Y_i,I_i)$ $(j \in J)$ of $\langle C,D\rangle$. Then, the authors of [27] utilize the fact that if $\langle A,B\rangle$ is a canonical generator of a new concept $\langle E,F\rangle$ and $\langle C,D\rangle$ is a non-canonical generator of $\langle E,F\rangle$ then any concept $\langle G,H\rangle$ such that $D \subset H$ and $B \not\subseteq H$ is not a canonical generator of any new concept, cf. [27, Proposition 1]. As a consequence, AddIntent does not have to process concepts like $\langle G,H\rangle$ during the search for canonical generators. On one hand, this is an analogous improvement like that proposed in this paper, see Lemma 2. On the other hand, improvements in both AddIntent and FCbO are based on different notions of canonicity (we do not use the lattice order $\leqslant$) and different approaches (incremental and non-incremental) of computing formal concepts.

4. Complexity and efficiency issues

It is a well-known fact that the limiting factor of computing all formal concepts is that the corresponding counting problem is #*P*-complete [18,20]. Fortunately, if $|I|$ is considerably small, one can get the set of all formal concepts in reasonable time even if X and Y are large. Therefore, there have been proposed various algorithms for FCA specialized on sparse incidence data. FCbO performs well in case of both sparse and dense data of reasonable size. From the point of view of the asymptotic worst-case complexity, FCbO has time delay $O(|Y|^3 \cdot |X|)$, see [14], and asymptotic time complexity $O(|\mathcal{B}(X,Y,I)| \cdot |Y|^2 \cdot |X|)$ because in the worst case, FCbO can degenerate into the original CbO [19,21] but in general, it cannot do worse than CbO. In addition, there are strong indications that on average FCbO delivers the results faster than CbO. Therefore, the average-case complexity analysis of FCbO and ramifications of the worst-case complexity of FCbO seem to be challenging and important *open problems*.

In this section we focus on two aspects of FCbO. First, we discuss suitable data representation for efficient computation of closures of $^{\downarrow\uparrow}$ which can be used by both CbO and FCbO including their derivatives like PCbO [15]. The second subsection presents an experimental evaluation of FCbO performance on real and synthesized data sets. The observations made therein illustrate the average-case behavior of the algorithm.

4.1. Improving efficiency of computing closures

The input formal context $I \subseteq X \times Y$ is usually represented in a computer by a two-dimensional array which corresponds with I in the obvious way. We suggest to represent I by a collection of sets representing all rows of such two-dimensional array. By a row (corresponding to $x \in X$) we mean a set of attributes $\{x\}^{\uparrow} = \{y \in I | \langle x,y\rangle \in I\}$. Clearly, if we take set $\mathcal{O} = \{\{x\}^{\uparrow} | x \in X\}$, we have $\langle x,y\rangle \in I$ iff $y \in \{x\}^{\uparrow}$ iff $x \in \{y\}^{\downarrow}$.

Representing I by $\mathcal{O}$, we can significantly improve the computation of a new formal concept which is done in lines 8 and 9 of Algorithm 2. Given formal concept $\langle A,B\rangle$ and $j \notin B$, we can compute

$$\langle C,D\rangle = \left\langle A \cap \{j\}^{\downarrow}, (A \cap \{j\}^{\downarrow})^{\uparrow} \right\rangle = \left\langle A \cap \{j\}^{\downarrow}, (B \cup \{j\})^{\downarrow\uparrow} \right\rangle$$

as it is shown in Algorithm 3. Thus, lines 8 and 9 of Algorithm 2 can be replaced by a single call of ComputeClosure($\langle A,B\rangle, j$). The algorithm is correct. Indeed, it is evident that $C = A \cap \{j\}^{\downarrow}$ and $D = \bigcap_{x \in A \cap \{j\}^{\downarrow}} \{x\}^{\uparrow} = \left(\bigcup_{x \in A \cap \{j\}^{\downarrow}} \{x\}\right)^{\uparrow} = (A \cap \{j\}^{\downarrow})^{\uparrow} = (B \cup \{j\})^{\downarrow\uparrow}$.

Algorithm 3: Procedure ComputeClosure($\langle A,B\rangle, j$)

```
1  set C to ∅;
2  set D to Y;
3  foreach x in A do
4    if j ∈ {x}↑ then
5      set C to C ∪ {x};
6      set D to D ∩ {x}↑;
7    end
8  end
9  return ⟨C,D⟩
```

Remark 3. A straightforward method for computing new formal concepts is based on definitions (5) and (6) of the concept-forming operators. The method can be implemented by a direct two-way algorithm which first computes the extent $(B \cup \{j\})^{\downarrow}$ which is further used to compute the intent $(B \cup \{j\})^{\downarrow\uparrow}$. Contrary to that, the procedure ComputeClosure computes the extent by filtering out the objects x from A for which it does not hold $j \in \{x\}^{\uparrow}$. In addition to that, during the computation of an extent, we also compute the corresponding intent by computing intersections of D and rows $\{x\}^{\uparrow}$. This can be done more efficiently especially if $\{x\}^{\uparrow}$ are organized as bit arrays. Thus, the algorithm relies on efficient implementation of sets and a single operation on sets: the intersection. Since computing intersections is generally more efficient than implementing the concept-forming operators, Algorithm 3 significantly outperforms the naive two-way algorithm. A detailed comparison of various data structures used for computing formal concepts can be found in [17]. ■

4.2. Experimental evaluation

We have run several experiments to compare the algorithm with CbO [19,21], Andrews's In-Close [2] and Ganter's NextClosure [9]. For the sake of comparison, we have implemented our algorithm, CbO and NextClosure in ANSI C while the implementation of In-Close was borrowed from the author. As suggested in the previous section, we represented input data tables by set $\mathcal{O}$ of table rows. Sets of attributes were represented by bit-arrays, where each bit represented the presence/absence of an attribute in a set. When storing a bit-array as an array of 32-bit or 64-bit integers, depending on the hardware architecture, all of the set operations with attributes, especially the set intersection, can be implemented by bitwise operations "and", "not", and "xor" on integers. These operations are implemented in arithmetic logic units (ALUs) of all computer processors. This representation is beneficial, e.g., in Algorithm 3 in line 6, where we can process up to 32 or 64 attributes at a time.

The experiments were run on otherwise idle 32-bit i386 hardware (Intel Core 2 Duo T9600, 2.8 GHz, 4 GB RAM). We performed two types of experiments. First, we were interested in the performance of all four algorithms measured by running time. Second, and more importantly, in order to evaluate the influence of the new canonicity test, we compared FCbO and CbO in terms of the total number of computed closures.

In the first set of experiments, we have run the algorithms on randomly generated data tables with various percentages of 1's in the table. We have used tables with 10,000 objects and the number of attributes ranging from 50 to 200 attributes. To illustrate the performance of algorithms, Fig. 4(left) shows a graph of dependency of time required to compute all formal concepts on the number of attributes in data tables with 10% of nonzero entries. We have not depicted the graph of average running time of NextClosure since there is a huge performance gap between the algorithm and the other algorithms (for instance FCbO is approximately 100 times faster than NextClosure on the evaluated data); the solid line is for FCbO, the dashed line is for CbO and the dotted line is for In-Close. In the FCbO/CbO comparison, the graph illustrating the average numbers of computed closures is depicted in Fig. 5(left); again, the solid line is for FCbO and the dashed line for CbO. Note that the graph

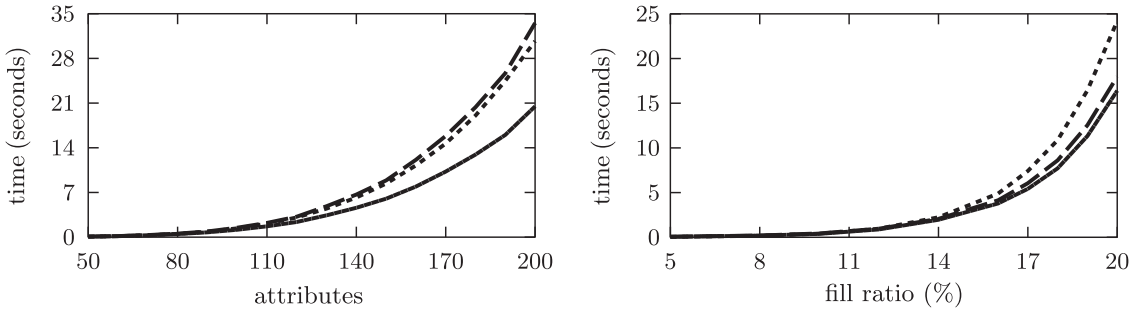

Fig. 4. Average running time dependent on number of attributes (on the left) and on fill ratio (density of 1's, on the right), solid line – FCbO, dashed line – CbO, dotted line – In-Close.

actually depicts the numbers divided by average concept lattice size (i.e., the number of closures which pass both the new, in case of FCbO, and the original canonicity test). Furthermore, to illustrate the influence of the fill ratio (density of 1's) of a data table on the speed of the algorithms and on the number of computed closures, we have included Figs. 4 and 5(right) which show graphs of dependencies on the fill ratios.

The second set of experiments were done with several data sets from the UCI Machine Learning Repository [4,13]. The results for performance times and numbers of computed closures are depicted in Fig. 6, along with the information on size and fill ratio of used data sets and concept lattice size.

From all the time and closure number dependency graphs and the table we can see that the FCbO algorithm significantly outperforms NextClosure and also considerably outperforms both the CbO and In-Close algorithms. In both cases, the performance gain is due to the new canonicity test which avoids a large number of concepts to be computed multiple times (cf. the numbers of closures computed by FCbO and CbO in case of the MUSHROOM data set, for instance, in Fig. 6). In case of NextClosure [9], the performance gain is then further multiplied by a more efficient computation of closures described in Section 4.1. The efficiency of the new "fast" test is illustrated by the graphs and the table depicting the numbers of closures computed by CbO (and NextClosure) and by FCbO.

5. Conclusions

We have introduced an algorithm called FCbO for computing formal concepts in object-attribute data tables. The algorithm results from CbO [19,21] by introducing a new canonicity test. We have proved correctness of the algorithm and pre-

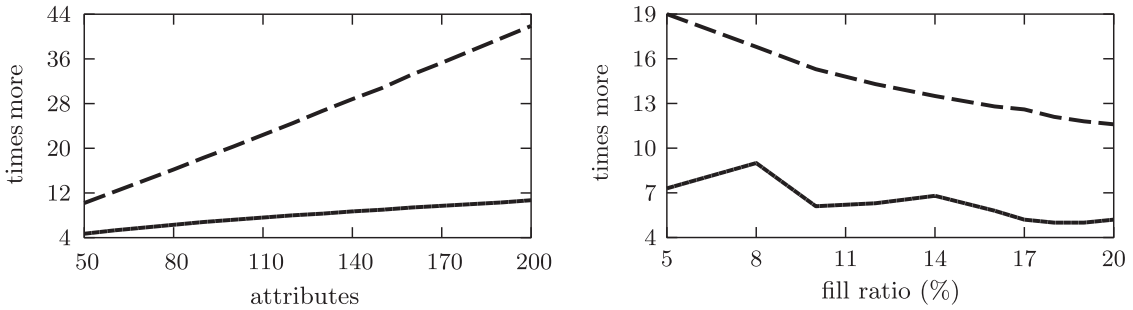

Fig. 5. Ratio of average number of closures computed by FCbO (solid line) and by CbO (dashed line) to average concept lattice size dependent on number of attributes (on the left) and on fill ratio (density of 1's, on the right).

dataset	mushroom	anonymous web	adult	internet ads
size	8124 × 119	32711 × 296	48842 × 104	3279 × 1557
fill ratio	19.33 %	1.02 %	8.65 %	0.88 %
#concepts	238710	129009	180115	9192
NextClosure time	53.891	243.325	134.954	114.493
CbO time	0.508	0.238	0.302	0.332
In-Close Time	0.544	0.524	0.302	0.158
FCbO time	0.340	0.240	0.318	0.160
#closures computed by CbO	1321411	785112	585253	1783871
#closures computed by FCbO	299202	398148	305644	309357

Fig. 6. Performance (in seconds) and numbers of closures computed by CbO and FCbO for selected datasets.

sented an experimental evaluation of its performance compared to the original CbO, Ganter's NextClosure and also to Andrews's In-Close, another contemporary derivative of CbO. The experiments have shown that FCbO significantly reduces the number of computed closures while maintaining a resonable overhead and hence delivers results faster than the other algorithms. The implementation of the algorithm can be downloaded from

http://fcalgs.sourceforge.net/fcbo-ins.html.

The future research will focus on further refinements and extensions of the algorithm and will focus in a more detail on the relationship between various recently-developed algorithms [2,27].

Acknowledgment

Supported by Grant Nos. P103/10/1056 and P202/10/P360 of the Czech Science Foundation and by Grant No. MSM 6198959214. The algorithm described in this paper has been presented during ICCS 2009 and CLA 2010 [16] conferences. However, in ICCS 2009, the algorithm took part in a performance contest only and was not published in the proceedings, and the CLA 2010 paper contains the pseudocode and a brief summary of the algorithm without further analysis or proofs. The present paper is aimed as a detailed explanation of the algorithm with emphasis on the canonicity test.

References

[1] R. Agrawal, T. Imielinski, A.N. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the ACM International Conference of Management of Data, 1993, pp. 207–216.
[2] S. Andrews, In-Close, a Fast Algorithm for Computing Formal Concepts, in: S. Rudolph, F. Dau, S.O. Kuznetsov (Eds.): *Supplementary Proceedings of ICCS '09*, CEUR WS, vol. 483, 2009, p. 14. <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-483/paper1.pdf>.
[3] F. Angiulli, E. Cesario, C. Pizzuti, Random walk biclustering for microarray data, Information Sciences 178 (6) (2008) 1479–1497.
[4] A. Asuncion, D. Newman, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2007.
[5] R. Belohlavek, Lattices of fixed points of fuzzy Galois connections, Math. Logic Quarterly 47 (1) (2001) 111–116.
[6] R. Belohlavek, E. Sigmund, J. Zacpal, Evaluation of IPAQ questionnaires supported by formal concept analysis, Inform. Sci. 181 (10) (2011) 1774–1786.
[7] R. Belohlavek, V. Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, J. Comput. Syst. Sci. 76 (1) (2010) 3–20.
[8] J.H. Correia, G. Stumme, R. Wille, U. Wille, Conceptual knowledge discovery – A human-centered approach, Appl. Artif. Intell. 17 (3) (2003) 281–302.
[9] B. Ganter, Two basic algorithms in concept analysis. (Technical Report FB4-Preprint No. 831). TH Darmstadt, 1984.
[10] B. Ganter, R. Wille, Formal Concept Analysis, Mathematical Foundations, Springer, Berlin, 1999.
[11] L.A. Goldberg, Efficient Algorithms for Listing Combinatorial Structures, Cambridge University Press, 1993.
[12] G.A. Gratzer, General Lattice Theory, 2nd ed., Birkhauser, 1998.
[13] S. Hettich, S.D. Bay, The UCI KDD Archive University of California, Irvine, School of Information and Computer Sciences, 1999.
[14] D.S. Johnson, M. Yannakakis, C.H. Papadimitriou, On generating all maximal independent sets, Inform. Process. Lett. 27 (3) (1988) 119–123.
[15] P. Krajca, J. Outrata, V. Vychodil, Parallel algorithm for computing fixpoints of galois connections, Ann. Math. Artif. Intell. 59 (2) (2010) 257–272.
[16] P. Krajca, J. Outrata, V. Vychodil, Advances in algorithms based on CbO, in: Proceedings of the CLA, 2010, pp. 325–337.
[17] P. Krajca, V. Vychodil, Comparison of data structures for computing formal concepts, in: Proc. MDAI, LNCS 5861, 2009, pp. 114–125.
[18] S.O. Kuznetsov, Interpretation on graphs and complexity characteristics of a search for specific patterns, Automat. Document. Math. Linguist. 24 (1) (1989) 37–45.
[19] S.O. Kuznetsov, A fast algorithm for computing all intersections of objects in a finite semi-lattice, Automat. Document. Math. Linguist. 27 (5) (1993) 11–21.
[20] S.O. Kuznetsov, On computing the size of a lattice and related decision problems, Order 18 (2001) 313–321.
[21] S.O. Kuznetsov, Learning of simple conceptual graphs from positive and negative examples, PKDD (1999) 384–391.
[22] S.O. Kuznetsov, S.A. Obiedkov, Comparing performance of algorithms for generating concept lattices, J. Exp. Theor. Artif. Int. 14 (2002) 189–216.
[23] W. Kneale, M. Kneale, The Development of Logic, Oxford University Press, USA, 1985.
[24] C. Lindig, Fast concept analysis, Working with Conceptual Structures – Contributions to ICCS 2000, Shaker Verlag, Aachen, 2000.
[25] H. Liu, X. Wang, J. He, J. Han, D. Xin, Z. Shao, Top–down mining of frequent closed patterns from very high dimensional data, Inform. Sci. 179 (7) (2009) 899–924.
[26] J. Medina, M. Ojeda-Aciego, Multi-adjoint t-concept lattices, Inform. Sci. 180 (5) (2010) 712–725.
[27] D. van der Merwe, S.A. Obiedkov, D.G. Kourie, AddIntent: A New Incremental Algorithm for Constructing Concept Lattices, in: Proceedings of the ICFCA 2004, LNAI 2961, 2004, pp. 205–206.
[28] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, H. Mannila, The Discrete Basis Problem, PKDD, Springer, 2006.
[29] B. Mirkin, Mathematical Classification and Clustering, Kluwer Academic Publishers, 1996.
[30] E.M. Norris, An algorithm for computing the maximal rectangles in a binary relation, Revue Roumaine de Mathématiques Pures et Appliquées 23 (2) (1978) 243–250.
[31] G. Snelting, F. Tip, Reengineering class hierarchies using concept analysis, ACM Trans. Program. Lang. Syst. 22 (3) (2000) 540–582.
[32] P. Tonella, Using a concept lattice of decomposition slices for program understanding and impact analysis, IEEE Trans. Softw. Eng. 29 (6) (2003) 495–509.
[33] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts, Ordered Sets, Dordrecht-Boston, 1982, pp. 445–470.
[34] M.J. Zaki, Mining non-redundant association rules, Data Min. Knowledge Discov. 9 (2004) 223–248.

Fundamenta Informaticae 115 (2012) 395–417
DOI 10.3233/FI-2012-661
IOS Press

Computing Formal Concepts by Attribute Sorting

Petr Krajca, Jan Outrata[*], Vilem Vychodil[†‡]
DAMOL: Data Analysis and Modeling Laboratory
Department of Computer Science
Palacky University, Olomouc, Czech Republic
vychodil@acm.org; jan.outrata@upol.cz; petr.krajca@binghamton.edu

Abstract. We present a novel approach to compute formal concepts of formal context. In terms of operations with Boolean matrices, the presented algorithm computes all maximal rectangles of the input Boolean matrix which are full of 1s. The algorithm combines basic ideas of previous approaches with our recent observations on the influence of attribute permutations and attribute sorting on the number of formal concepts which are computed multiple times. As a result, we present algorithm which computes formal concepts by successive context reduction and attribute sorting. We prove its soundness, discuss its complexity and efficiency, and show that it outperforms other algorithms from the CbO family in terms of substantially lower numbers of formal concepts which are computed multiple times.

1. Introduction and Problem Setting

Formal concept analysis (FCA) is a method of relational data analysis proposed by R. Wille [27] in early 80's. Since its inception, there has been an extensive theoretical research which has lead to many order-theoretical results, see [7] for a survey. Another, maybe equally important fact is that the results have been directly applied to various fields of data analysis including analysis in software engineering [25, 26], web information retrieval [11], and market-basket analysis [29]. Examples of FCA applications can be found in [4, 7].

[*]Jan Outrata was supported by grant no. P202/10/P360 of the Czech Science Foundation.
[†]Petr Krajca and Vilem Vychodil were supported by grant no. P103/10/1056 of the Czech Science Foundation.
[‡]Address for correspondence: Department of Computer Science, Palacky University, 17. listopadu 12, CZ–77146 Olomouc, Czech Republic

In its basic setting, FCA deals with object-attribute relational data which can be seen as a data table with rows corresponding to objects, columns corresponding to attributes (features), and table entries being 1 or 0, indicating whether objects have or do not have corresponding attributes. Formally, such data tables can be seen as binary relations between a set of objects and a set of attributes. The aim of FCA is to extract from such input data useful information about interesting object-attribute biclusters and attribute dependencies which are present in data. The outputs of FCA are used either directly or for preprocessing purposes. In the first case, extracted object-attribute clusters (so-called formal concepts) are ordered by a subconcept-superconcept hierarchy and can be presented to users by a line diagram of clusters (diagram of so-called concept lattice). The users can then navigate though the hierarchy to find clusters, identified by sets of objects and attributes that are covered by the clusters, which represent interesting and/or useful information for them. For instance, in an object-attribute database of cars and their features, users can find clusters like "affordable and safe cars", "four-wheel drive SUVs", etc., which they may find interesting. Note that the interpretation of a cluster as a concept having its extent (objects that fall under the concept) and its intent (attributes that fall under the concept) which is used in FCA is inspired by a traditional understanding of concept which goes back to Port-Royal logic [5, 18].

If FCA is used for preprocessing, the extracted clusters (formal concepts) are not used by users directly. Instead, they are used as input for other data mining methods. For instance, the seminal paper [24] showed that the formal concepts can be used to find non-redundant association rules, cf. also [29]. Recently, it has been shown in [3] that formal concepts can be used to find optimal factorization of Boolean matrices. In fact, it can be shown that they correspond to optimal solutions of the discrete basis problem discussed by Miettinen et al. [21].

In either case, the basic computational problem of FCA is to compute, given an input formal context (an object-attribute data table), the set of all formal concepts (the object-attribute clusters present in the input data). In the past, there have been proposed various algorithms for solving this task, see [17] for a survey and comparison. Among the best-known algorithms are CbO [14, 15, 16] proposed by Kuznetsov, Ganter's NextClosure [6, 7], and Lindig's UpperNeighbor [19] algorithms. There is an important family of algorithms which includes CbO, NextClosure, the algorithm proposed by Norris [22], and other algorithms such as PCbO [12], FCbO [13, 23], and InClose [2]. We call this family a *CbO family* because all algorithms in the family can be seen as modifications or refinements of CbO. For instance, NextClosure can be seen an iterative version of CbO, PCbO is a parallel variant of CbO, FCbO is a refinement of CbO which uses a new canonicity test, etc. In a broader sense, the CbO family of algorithms can be seen as an example of a family of algorithms for listing combinatorial structures [8].

A common issue that all algorithms for FCA have to care about is to prevent processing (e.g., storing or listing) the same formal concept multiple times. There are several approaches to cope with the problem. The CbO family algorithms use canonicity tests which are generally very cheap to perform. The basic idea is the following. Formal concepts are supposed to be computed in a predefined order. If the order is not preserved in a certain branch of computation (i.e., a newly computed formal concept does not pass the canonicity test during the computation), the branch is no longer considered. As a consequence, the canonicity test ensures that even if a formal concept is computed several times, it is processed (e.g., stored or listed) exactly once.

Although conceptually similar, algorithms from the CbO family differ in their efficiency. One of the most important factors is just the efficiency of the underlying canonicity tests. For instance, FCbO

uses a canonicity test which is more efficient than that of the original CbO. In practice, the numbers of formal concepts which are computed multiple times by FCbO is considerably smaller than the numbers corresponding to CbO [13, 23]. Another efficiency issue which is related to canonicity tests is the order in which attributes are processed by algorithms of the CbO family. In general, an important feature of algorithms for FCA is whether their performance depends on the order of objects and attributes in the input formal context. From this point of view, we shall call an algorithm (*permutation*) *resistant* whenever all isomorphic copies (in the usual sense) of the input formal context require the same number of elementary computation steps in order to compute all concepts. For our purposes, an elementary computation step shall be represented by computation of a single formal concept. One can easily see that, e.g., Lindig's UpperNeighbor algorithm [19] is resistant. In other words, if we rearrange rows and columns in the input data table, the algorithm uses the exact same number of steps to compute all formal concepts. On the other hand, algorithms from the CbO family are not resistant [13] and thus considering different orders of attributes can reduce the number of concepts that are computed multiple times, thus improving the efficiency.

The present paper is partly motivated by our observations from [13] where we have investigated the impact of using different orders of attributes for algorithms from the CbO family. One of the results presented in [13] says that if attributes of formal context are sorted in the ascending order according to their supports, i.e., the numbers of objects having the attributes, then the canonicity test of both CbO and FCbO always succeeds for all attribute concepts (concepts generated by a single attribute) provided that all attributes are distinct (i.e., all columns of the input data table are pairwise distinct). Furthermore, our empirical experiments have shown an interesting tendency that while processing formal contexts with attributes sorted in the aforementioned order, canonicity tests tend to fail less frequently than in the case of contexts containing inversions (with respect to the aforementioned order). In addition, with increasing number of inversions in a data table, the average number of computed closures grows. This seems to be a general tendency which has been experimentally observed in [13].

In the present paper, we elaborate on the ideas of attribute sorting. Motivated by the results of attribute sorting presented in [13], we introduce a method for attribute sorting and context reduction which is performed after obtaining a new formal concept. Unlike the approach in [13], where attribute sorting was just a means of data preprocessing and was used for each input data exactly once (before the computation which is then done by standard CbO or FCbO), we utilize attribute sorting during the computation several times which results in a conceptually new algorithm. The idea of dynamic reordering of attributes appeared in algorithm CHARM [28] for computing closed itemsets. In the paper, we describe the algorithm, prove its soundness, and investigate its complexity and further efficiency issues related to efficiency of its canonicity test. As we shall see in further sections, in terms of the numbers of concepts computed multiple times, the proposed algorithm outperforms CbO by an order of magnitude. The improvement is apparent especially in the case of large real data sets [9].

The paper is organized as follows. Section 2 contains brief preliminaries from FCA. Section 3 introduces operations with formal contexts which are used to describe the algorithm. Section 4 introduces the algorithm. Section 5 contains a detailed running example of the algorithm. Section 6 contains proof of soundness of the algorithm. Finally, Section 7 is devoted to complexity and efficiency issues of the algorithm and contains performance comparison with other algorithm from the CbO family.

2. Preliminaries from FCA

In this section we recall basic notions of FCA. More details can be found in monographs [7] and [4]. Let X and Y denote finite sets of objects and attributes, respectively. A formal context is a triple $\mathbb{K} = \langle X, Y, I \rangle$ where $I \subseteq X \times Y$, i.e. I is a binary relation between X and Y. The fact $\langle x, y \rangle \in I$ is interpreted so that "object x has attribute y". Note that $\mathbb{K}$ obviously corresponds to a two-dimensional data table with rows corresponding to objects from X, columns corresponding to attributes from Y, and table entries being 1 and 0 indicating whether $\langle x, y \rangle \in I$ or $\langle x, y \rangle \notin I$. Thus, formal contexts can be seen as Boolean matrices.

Given $\mathbb{K} = \langle X, Y, I \rangle$, we introduce a pair of concept-forming operators [7] ${}^{\uparrow_{\mathbb{K}}} : 2^X \to 2^Y$ and ${}^{\downarrow_{\mathbb{K}}} : 2^Y \to 2^X$ defined, for each $A \subseteq X$ and $B \subseteq Y$, by $A^{\uparrow_{\mathbb{K}}} = \{y \in Y \mid \text{for each } x \in A : \langle x, y \rangle \in I\}$ and $B^{\downarrow_{\mathbb{K}}} = \{x \in X \mid \text{for each } y \in B : \langle x, y \rangle \in I\}$, respectively. If there is no danger of confusion, we omit $\mathbb{K}$ and write just ${}^{\uparrow}$ and ${}^{\downarrow}$ instead of ${}^{\uparrow_{\mathbb{K}}}$ and ${}^{\downarrow_{\mathbb{K}}}$, respectively. The cardinality of $\{y\}^{\downarrow_{\mathbb{K}}}$ is called the support of $y \in Y$. By a formal concept in $\mathbb{K}$ with extent A and intent B we mean any pair $\langle A, B \rangle \in 2^X \times 2^Y$ such that $A^{\uparrow_{\mathbb{K}}} = B$ and $B^{\downarrow_{\mathbb{K}}} = A$. Thus, formal concepts are fixed points of the concept-forming operators. Intuitively, each formal concept $\langle A, B \rangle$ represents a bicluster in $\mathbb{K}$ which consists of objects A that fall under the concept and attributes B that fall under the concept. Since $A^{\uparrow_{\mathbb{K}}} = B$ and $B^{\downarrow_{\mathbb{K}}} = A$, A is a set of objects having all attributes from B and B is a set of attributes shared by all objects from A. Let us stress that formal concepts can be seen as maximal Boolean submatrices in the following sense: any $\langle A, B \rangle \in 2^X \times 2^Y$ such that $A \times B \subseteq I$ can be called a Boolean submatrix of $\mathbb{K}$ (which is full of 1s). Moreover, a Boolean submatrix $\langle A, B \rangle$ of $\mathbb{K}$ is a maximal one if, for each Boolean submatrix $\langle A', B' \rangle$ of $\mathbb{K}$ such that $A \times B \subseteq A' \times B'$, we have $A = A'$ and $B = B'$. We have that $\langle A, B \rangle \in 2^X \times 2^Y$ is a maximal Boolean submatrix of $\mathbb{K}$ (which is full of 1s) iff $A^{\uparrow_{\mathbb{K}}} = B$ and $B^{\downarrow_{\mathbb{K}}} = A$. Hence, maximal Boolean submatrices full of 1s are exactly the formal concepts.

The set of all formal concepts in $\mathbb{K} = \langle X, Y, I \rangle$ will be denoted by $\mathcal{B}(X, Y, I)$. Recall that $\mathcal{B}(X, Y, I)$ endowed by a concept ordering $\leq$ forms a complete lattice, called a concept lattice, whose structure is described by the Basic Theorem of FCA [7, 27].

3. Clarification and Attribute Sorting

In this section, we introduce basic operations with contexts that are used to describe the proposed algorithm for computing formal concepts. One of the distinguishing features of the algorithm is that during the computation, it transforms an initial formal context into other contexts by taking subsets of objects and by grouping several attributes together. In addition to that, groups of attributes are sorted according to their support and equipped with an additional numerical flag indicating whether a group of attributes is allowed to be present in intents of formal concepts computed in next stages (a precise meaning of the flag will be described later). These operations on contexts play a crucial role and will be described in this section. We begin with particular representation of formal contexts.

3.1. Input Formal Contexts and R-contexts

Here we describe the basic form of formal concepts which are used during the computation. As in case of any algorithm for computing formal concepts, the input for our algorithm is a formal context

$\mathbb{K} = \langle X, Y, I \rangle$. In order to keep information about groups of attributes, we use particular contexts, called R-contexts, to represent input data. A formal definition follows.

Definition 3.1. Given a formal context $\mathbb{K} = \langle X, Y, I \rangle$, a triple $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp \rangle$ is called an *R-context* (*derived from* $\mathbb{K}$) if the following conditions are satisfied:

(i) $X^\sharp \subseteq X$;

(ii) $Y^\sharp \subseteq \mathbb{N}_0 \times 2^Y$ such that for any $\langle n_1, B_1 \rangle \in Y^\sharp$ and $\langle n_2, B_2 \rangle \in Y^\sharp$ we have either that (a) $n_1 = n_2$ and $B_1 = B_2 \neq \emptyset$ or (b) $B_1 \neq \emptyset$, $B_2 \neq \emptyset$, and $B_1 \cap B_2 = \emptyset$;

(iii) for any $x \in X^\sharp$ and $\langle n, B \rangle \in Y^\sharp$: $\langle x, y_1 \rangle \in I$ iff $\langle x, y_2 \rangle \in I$ holds true for all $y_1, y_2 \in B$;

(iv) $I^\sharp = \{\langle x, \langle n, B \rangle\rangle \in X^\sharp \times Y^\sharp \mid \langle x, y \rangle \in I \text{ for all } y \in B\}$.

In addition, $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp \rangle$ is called an *initial R-context* (*derived from* $\mathbb{K}$) if $X^\sharp = X$, $Y^\sharp = \{\langle 0, \{y\}\rangle \mid y \in Y\}$, and $I^\sharp = \{\langle x, \langle 0, \{y\}\rangle\rangle \in X^\sharp \times Y^\sharp \mid \langle x, y \rangle \in I\}$. ■

We can immediately observe basic properties of R-contexts:

Remark 3.2. (a) Each R-context is a formal context. Notice that due to (iv), $\langle x, \langle n, B \rangle\rangle \in I^\sharp$ iff $x \in B^{\downarrow_{\mathbb{K}}}$ for $x \in X^\sharp$ and $\langle n, B \rangle \in Y^\sharp$. Moreover, taking into account (iii) and (iv), it follows that for any $\langle x, \langle n, B \rangle\rangle \in X^\sharp \times Y^\sharp$, $\langle x, \langle n, B \rangle\rangle \in I^\sharp$ iff there is $y \in B$ such that $\langle x, y \rangle \in I$ in which case $\langle x, y \rangle \in I$ is true for all $y \in B$ because of (iii). Note that each attribute $\langle n, B \rangle \in Y^\sharp$ has two parts: a numerical flag n (explained later) and a subset $B \subseteq Y$ of original attributes. Using (ii), we get that $B \neq \emptyset$. In addition to that, distinct attributes from $Y^\sharp$ have associated pairwise disjoint nonempty subsets of original attributes.

(b) Note that attributes in R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp \rangle$ have natural interpretation as sets of attributes from the original context which are indistinguishable in $\mathbb{K}$ provided we restrict ourselves only to objects from $X^\sharp$. Indeed, this is a basic consequence of Definition 3.1 (iii).

(c) An initial R-context derived from $\mathbb{K}$ is an R-context. Indeed, (i) and (ii) are obvious since attributes of an initial R-context are all of the form $\langle 0, \{y\}\rangle$. It is immediate that (iii) and (iv) of Definition 3.1 are satisfied as well. Obviously, an initial R-context $\mathbb{K}^\sharp$ derived from $\mathbb{K}$ is isomorphic to $\mathbb{K}$ in the usual sense. In other words, $\mathbb{K}^\sharp$ is exactly the same as $\mathbb{K}$ up to the names of attributes.

From now on, we describe further operations with contexts in terms of R-contexts instead of the original input contexts. By this we do not impose any restriction since an initial R-context derived from $\mathbb{K}$ has the same concepts up to different names of attributes, see Remark 3.2 (c).

Example 3.3. As an example, we consider a formal context $\mathbb{K}$ with objects $X = \{a, \ldots, f\}$ and attributes $Y = \{0, \ldots, 7\}$. The context (left) and an R-context derived from $\mathbb{K}$ (right) are depicted in Table 1. Notice that the original attributes 1 and 4 are distinguishable in $\mathbb{K}$ by object c. On the other hand, they are indistinguisbahle on $\{b, d, e, f\}$, hence the attribute $\langle 0, \{1, 4\}\rangle$ in $Y^\sharp$ is correct and satisfies the requirement given by Definition 3.1 (iii). Also, note that all attributes in $\mathbb{K}^\sharp$ except for $\langle 1, \{2\}\rangle$ are given zero flags.

Remark 3.4. Note that $\mathbb{K}^\sharp$ which results from $\mathbb{K}$ is fully given by the sets $X^\sharp$ and $Y^\sharp$ of objects and attributes, respectively. The binary relation $I^\sharp$ can be determined from the original I, see Remark 3.2 (a). Thus, a concise computer representation of $\mathbb{K}^\sharp$ can consist of a list of objects and attributes, respectively,

Table 1. Formal context $\mathbb{K}$ (left) and an R-context derived from $\mathbb{K}$ (right)

$\mathbb{K}$	0	1	2	3	4	5	6	7
a		×	×		×			
b	×		×	×		×		×
c				×	×			×
d	×	×	×		×	×	×	×
e	×	×			×	×		
f	×		×	×		×		×

$\mathbb{K}^\sharp$	$\langle 0,\{1,4\}\rangle$	$\langle 1,\{2\}\rangle$	$\langle 0,\{3\}\rangle$	$\langle 0,\{6\}\rangle$	$\langle 0,\{7\}\rangle$
b		×	×		×
d	×	×		×	×
e	×				
f		×	×		×

omitting the expensive operation of copying a part of the data representation of I which can be kept in computer memory only once.

We conclude this subsection by showing that the concept-forming operators induced by R-contexts have a close relationship to concept-forming operators of original contexts. In order to keep concise notation, we first introduce the following abbreviation. For any $D \subseteq Y^\sharp$, we define

$$\lfloor D \rfloor = \bigcup\{B \subseteq Y \mid \langle n, B\rangle \in D\}. \tag{1}$$

Using this notation, we have:

Lemma 3.5. Let $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp\rangle$ be an R-context derived from $\mathbb{K}$. Then, for any $C \subseteq X^\sharp$ and $D \subseteq Y^\sharp$,

$$\lfloor C^{\uparrow_{\mathbb{K}^\sharp}} \rfloor = C^{\uparrow_{\mathbb{K}}} \cap \lfloor Y^\sharp \rfloor, \tag{2}$$

$$X^\sharp \cap \lfloor D \rfloor^{\downarrow_{\mathbb{K}}} = D^{\downarrow_{\mathbb{K}^\sharp}}. \tag{3}$$

Proof:
Both equalities can be proved using basic properties of R-contexts.

"(2)": Let $y \in \lfloor C^{\uparrow_{\mathbb{K}^\sharp}} \rfloor$. Therefore, there is $\langle n, B\rangle \in C^{\uparrow_{\mathbb{K}^\sharp}} \subseteq Y^\sharp$ such that $y \in B$. Hence, $y \in \lfloor Y^\sharp \rfloor$. Moreover, $\langle n, B\rangle \in C^{\uparrow_{\mathbb{K}^\sharp}}$ yields that for each $x \in C$, $\langle x, \langle n, B\rangle\rangle \in I^\sharp$. Due to Definition 3.1 (iv), the latter means that for each $y \in B$ and $x \in C$, we have $\langle x, y\rangle \in I$. Therefore, $B \subseteq C^{\uparrow_{\mathbb{K}}}$, i.e., $y \in C^{\uparrow_{\mathbb{K}}}$, showing $\lfloor C^{\uparrow_{\mathbb{K}^\sharp}} \rfloor \subseteq C^{\uparrow_{\mathbb{K}}} \cap \lfloor Y^\sharp \rfloor$. Conversely, take $y \in C^{\uparrow_{\mathbb{K}}} \cap \lfloor Y^\sharp \rfloor$. Then, for each $x \in C$, we have that $\langle x, y\rangle \in I$. Since $y \in \lfloor Y^\sharp \rfloor$, there is $\langle n, B\rangle \in Y^\sharp$ such that $y \in B$. Since $C \subseteq X^\sharp$, using Definition 3.1 (iii) and the previous fact, we get that for each $y \in B$ and $x \in C$, $\langle x, y\rangle \in I$. Thus, for each $x \in C$, $\langle x, \langle n, B\rangle\rangle \in I^\sharp$, meaning $\langle n, B\rangle \in C^{\uparrow_{\mathbb{K}^\sharp}}$ and thus $y \in B \subseteq \lfloor C^{\uparrow_{\mathbb{K}^\sharp}} \rfloor$.

"(3)": Consider $x \in X^\sharp \cap \lfloor D \rfloor^{\downarrow_{\mathbb{K}}}$. Therefore, for each $y \in \lfloor D \rfloor$, $\langle x, y\rangle \in I$. In particular, for any $B \subseteq \lfloor D \rfloor$ such that $\langle n, B\rangle \in D$, we have $\langle x, y\rangle \in I$ for all $y \in B$, i.e., $\langle x, \langle n, B\rangle\rangle \in I^\sharp$ since $x \in X^\sharp$. Moreover, $\langle n, B\rangle \in D$ has been taken arbitrarily, which means that $x \in D^{\downarrow_{\mathbb{K}^\sharp}}$. Conversely, let $x \in D^{\downarrow_{\mathbb{K}^\sharp}}$. By definition, we have $\langle x, \langle n, B\rangle\rangle \in I^\sharp$ for all $\langle n, B\rangle \in D$. Therefore, $\langle x, y\rangle \in I$ for all $y \in B$ such that $\langle n, B\rangle \in D$, meaning that $\langle x, y\rangle \in I$ is true for all $y \in \lfloor D \rfloor$. Therefore, $x \in \lfloor D \rfloor^{\downarrow_{\mathbb{K}}}$. The fact that $x \in X^\sharp$ is trivial. □

3.2. Clarification

Each R-context can be transformed into a new R-context with possibly smaller sets of attributes by a process of clarification. Recall from [7] that a formal context $\mathbb{K} = \langle X, Y, I \rangle$ is called clarified if for any $y_1, y_2 \in Y$ it follows that $\{y_1\}^{\downarrow} = \{y_2\}^{\downarrow}$ implies $y_1 = y_2$ and dually for any couple of objects. In other words, a clarified context in sense of [7] is a formal context where all columns in the corresponding object-attribute data table are distinct and dually for rows. It is a well known fact that taking a clarified formal context (with duplicate rows and columns removed) instead of the original one we get a possibly smaller context whose concept lattice is isomorphic to the concept lattice of the original context.

In this section, we focus on a particular clarification of R-contexts which applies only to attributes of R-contexts. In addition, the procedure of clarification we introduce here produces an R-context as a result, i.e., we cope with particular form of attributes which consist of a flag and a set of attributes of the original context, see Definition 3.1. The basic idea is the same as in [7], we produce a new R-context by putting together identical columns of the corresponding data table.

For any R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp \rangle$ which is derived from $\mathbb{K}$, we can consider a binary relation $\equiv_{\mathbb{K}^\sharp}$ on $Y^\sharp$ such that $y_1 \equiv_{\mathbb{K}^\sharp} y_2$ iff $\{y_1\}^{\downarrow_{\mathbb{K}^\sharp}} = \{y_2\}^{\downarrow_{\mathbb{K}^\sharp}}$. Hence, $y_1 \equiv_{\mathbb{K}^\sharp} y_2$ if columns of the data table corresponding to $\mathbb{K}^\sharp$ given by attributes y_1 and y_2 are the same. Obviously, $\equiv_{\mathbb{K}^\sharp}$ is an equivalence relation and thus we may consider the corresponding quotient set $Y^\sharp / \equiv_{\mathbb{K}^\sharp}$, denoting the equivalence class of $\equiv_{\mathbb{K}^\sharp}$ containing $y \in Y^\sharp$ by $[y]_{\equiv_{\mathbb{K}^\sharp}}$. Under this notation, we introduce the following notion:

Definition 3.6. For any R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp \rangle$, we define $X^\complement$, $Y^\complement$, $I^\complement$ as follows:

(i) $X^\complement = X^\sharp$;

(ii) $Y^\complement = \big\{ \big\langle \sum\{n \in \mathbb{N}_0 \mid \langle n, B \rangle \in [y]_{\equiv_{\mathbb{K}^\sharp}}\}, \big\lfloor [y]_{\equiv_{\mathbb{K}^\sharp}} \big\rfloor \big\rangle \mid y \in Y^\sharp \big\}$,

(iii) $I^\complement = \big\{ \langle x, \langle n, B \rangle \rangle \in X^\complement \times Y^\complement \mid$ there is $n' \leq n$ and $B' \subseteq B$ such that $\langle x, \langle n', B' \rangle \rangle \in I^\sharp \big\}$.

Moreover, $\mathbb{K}^\complement = \langle X^\complement, Y^\complement, I^\complement \rangle$ is called a *clarified R-context* (*which results from* $\mathbb{K}^\sharp$). ■

Remark 3.7. Examining (ii) of Definition 3.6, the set $Y^\complement$ of attributes contains pairs $\langle n, B \rangle$, where n is a numerical flag which results by taking a sum of flags of all attributes in a single equivalence class of $\equiv_{\mathbb{K}^\sharp}$. Analogously, B is a union of sets of original attributes which can be found in attributes from the same equivalence class. While the idea behind taking unions of sets of attributes is clear since attributes indistinguishable under $\equiv_{\mathbb{K}^\sharp}$ are grouped together, the intuitive meaning of taking a sum of flags may not be clear at this point. The informal explanation is the following: in $\langle n, B \rangle \in Y^\sharp$, the number n says that "exactly n of the original attributes from B are not permitted to be used (at certain level of computation)". Thus, if we group attributes together, the numbers of attributes which are not permitted are added since the sets of attributes are disjoint. A formal justification will follow in Section 4.

The following assertion shows basic properties of clarified R-contexts.

Lemma 3.8. Each clarified R-context $\mathbb{K}^\complement$ is a well-defined R-context. Moreover, for $\mathbb{K}^\complement$ we have that $\equiv_{\mathbb{K}^\complement}$ is identity. As a consequence, $(\mathbb{K}^\complement)^\complement = \mathbb{K}^\complement$.

Proof:
It suffices to check requirements (ii)–(iv) of Definition 3.1. It is immediate that (ii) is satisfied since $Y^\sharp/{\equiv_{\mathbb{K}^\sharp}}$ consists of pairwise disjoint and nonempty classes which define attributes in $\mathbb{K}^\complement$. Furthermore, (iii) is satisfied because for any $x \in X^\complement = X^\sharp$, $\langle n, B\rangle \in Y^\complement$, and $y_1, y_2 \in B$, there are $\langle n_1, B_1\rangle \in Y^\sharp$ and $\langle n_2, B_2\rangle \in Y^\sharp$ such that $y_1 \in B_1$, $y_2 \in B_2$, and $\langle n_1, B_1\rangle \equiv_{\mathbb{K}^\sharp} \langle n_2, B_2\rangle$. Therefore, $\langle x, y_1\rangle \in I$ iff $\langle x, \langle n_1, B_1\rangle\rangle \in I^\sharp$ iff $\langle x, \langle n_2, B_2\rangle\rangle \in I^\sharp$ iff $\langle x, y_2\rangle \in I$, i.e. (iii) is satisfied by $\mathbb{K}^\complement$. In order to show (iv), observe that by Definition 3.6, $\langle x, \langle n, B\rangle\rangle \in I^\complement$ iff there is $n' \leq n$ and $B' \subseteq B$ such that $\langle n', B'\rangle \in Y^\sharp$ and $\langle x, \langle n', B'\rangle\rangle \in I^\sharp$. Taking into account $\equiv_{\mathbb{K}^\sharp}$, $\langle x, \langle n, B\rangle\rangle \in I^\complement$ iff for any $\langle n', B'\rangle \in Y^\sharp$ such that $n' \leq n$ and $B' \subseteq B$, we have $\langle x, \langle n', B'\rangle\rangle \in I^\sharp$. Using Definition 3.1 (iv) which holds for $\mathbb{K}^\sharp$, the latter is true iff for any $\langle n', B'\rangle \in Y^\sharp$ such that $n' \leq n$ and $B' \subseteq B$, we have $\langle x, y\rangle \in I$ for all $y \in B'$, i.e., $\langle x, y\rangle \in I$ for all $y \in B$ because B is a union of all such B's, proving (iv) of Definition 3.1 for $\mathbb{K}^\complement$. The remaining claims follow easily. □

Table 2. Clarified R-context $\mathbb{K}^\complement$ (left) and $\mathbb{K}^\complement$ with sorted attributes (right).

$\mathbb{K}^\complement$	$\langle 0,\{1,4\}\rangle$	$\langle 1,\{2,7\}\rangle$	$\langle 0,\{3\}\rangle$	$\langle 0,\{6\}\rangle$
b		×	×	
d	×	×		×
e	×			
f		×	×	

$\mathbb{K}^\complement$	$\langle 0,\{6\}\rangle$	$\langle 0,\{1,4\}\rangle$	$\langle 0,\{3\}\rangle$	$\langle 1,\{2,7\}\rangle$
b			×	×
d	×	×		×
e		×		
f			×	×

Example 3.9. Consider $\mathbb{K}$ and $\mathbb{K}^\sharp$ from Table 1. Then, the clarified R-context which results from $\mathbb{K}^\sharp$ is depicted in Table 2. Notice that only original attributes that have been put together are $\langle 1,\{2\}\rangle$ and $\langle 0,\{7\}\rangle$. Since the flags are added, the flag of the resulting attribute $\langle 1,\{2,7\}\rangle$ is equal to 1. Flags of the other attributes remain zero.

Remark 3.10. Notice that in our approach, we do not consider clarification of objects, i.e., $\mathbb{K}^\complement$ may contain several objects having the same attributes. Clarification of objects is not used in the subsequent algorithm because in our approach it would not reduce the number of concepts computed multiple times and is therefore omitted.

3.3. Attribute Sorting

The algorithm described in Section 4 relies on attribute sorting. In particular, for each R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp\rangle$, we consider a partial order $\leq^\sharp$ on $Y^\sharp$ such that for any $y_1, y_2 \in Y^\sharp$, $y_1 \leq^\sharp y_2$ implies $|\{y_1\}^{\downarrow_{\mathbb{K}^\sharp}}| \leq |\{y_2\}^{\downarrow_{\mathbb{K}^\sharp}}|$. In general, $\leq^\sharp$ is not a linear order (not even in the case of clarified R-contexts) but it can be extended to a linear order by a well-known procedure of topological sorting.

In next sections, we do not use $\leq^\sharp$ directly. Instead, we assume that we have a bijective map which assigns to each attribute from $Y^\sharp$ its numerical index which represents a position in an ordered list of attributes which are sorted according to (a linear extension of) $\leq^\sharp$. In a more detail, for any R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp\rangle$ we consider a bijective map $f\colon Y^\sharp \to \{0, \ldots, |Y^\sharp|-1\}$ such that, for any $y_1, y_2 \in Y^\sharp$,

$$\text{if } f(y_1) \leq f(y_2), \text{ then } |\{y_1\}^{\downarrow_{\mathbb{K}^\sharp}}| \leq |\{y_2\}^{\downarrow_{\mathbb{K}^\sharp}}|. \tag{4}$$

The inverse f^{-1} of f is a map which assigns to each index $j \in \{0, \ldots, |Y^\sharp| - 1\}$ the corresponding attribute $f^{-1}(j) \in Y^\sharp$.

Example 3.11. Any R-context can be depicted with attributes sorted according to f. That is, if $f(y_1) < f(y_2)$, then y_1 is depicted before y_2. Table 2 (right) shows the results if we apply this idea to the R-context $\mathbb{K}^\complement$ from Table 2 (left). Note that in this particular case, there are two ways to define f since attributes $\langle 0, \{1, 4\}\rangle$ and $\langle 0, \{3\}\rangle$ have the same support. In such situations, we always consider an arbitrary (but fixed) f for the same R-context.

Remark 3.12. In [13], we have investigated the influence of attribute sorting for the CbO family of algorithms. From this point of view, we have considered the same ordering of attributes according to their support. An important distinguishing feature of the present approach is that we do not consider single $\leq^\sharp$ (i.e., a single f) during the computation. Instead, during the computation, we successively reduce the initial R-context and after each reduction, we determine new f which applies to the reduced R-context.

3.4. Context Reduction

We now describe a particular reduction operation on R-contexts which utilizes operations on R-contexts defined in previous sections. The algorithm described in Section 4 uses this operation directly to reduce the problem of computing formal concepts of an R-context to the problem of computing formal concepts of several smaller R-contexts. From this point of view, the proposed algorithm follows the usual *divide et impera* scheme of decomposing an instance of a problem into several instances of the same problem of smaller sizes which in turn leads to a concise implementation of the algorithm by a recursive procedure.

The input for reduction is a clarified R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp\rangle$ and a formal concept $\langle C, D\rangle$ in $\mathbb{K}^\sharp$ whose intent is nonempty, i.e. $D \neq \emptyset$. For $\mathbb{K}^\sharp$, we assume that we are given a bijective map satisfying (4) which determines the order of attributes in $\mathbb{K}^\sharp$. Since D is nonempty, we can denote by $\min(D)$ the least attribute from D with respect to the order given by f, i.e., $\min(D) \in D$ such that $f(\min(D)) \leq f(y)$ for all $y \in D$. Using this notation, we define the following notion:

Definition 3.13. For any R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp\rangle$, $C \subseteq X^\sharp$, $\emptyset \neq D \subseteq Y^\sharp$ such that $C^{\uparrow_{\mathbb{K}^\sharp}} = D$ and $D^{\downarrow_{\mathbb{K}^\sharp}} = C$, we define $X^\mathfrak{R}$, $Y^\mathfrak{R}$, and $I^\mathfrak{R}$ as follows:

(i) $X^\mathfrak{R} = C$;

(ii) $Y^\mathfrak{R} = \{\mathrm{Attr}(y) \,|\, y \in Y^\sharp \text{ and } y \notin D\}$, where $\mathrm{Attr}(y) \in \mathbb{N}_0 \times 2^Y$ is defined by

$$\mathrm{Attr}(\langle n, B\rangle) = \begin{cases} \langle |B|, B\rangle, & \text{if } n = 0 \text{ and } f(\langle n, B\rangle) < f(\min(D)), \\ \langle n, B\rangle, & \text{otherwise,} \end{cases} \tag{5}$$

for any $\langle n, B\rangle \in Y^\sharp$;

(iii) $I^\mathfrak{R} = \big\{\langle x, \langle n, B\rangle\rangle \in X^\mathfrak{R} \times Y^\mathfrak{R} \,|\, \text{there is } n' \leq n \text{ such that } \langle x, \langle n', B\rangle\rangle \in I^\sharp\big\}$. ■

Remark 3.14. One can easily see that $\mathbb{K}^\mathfrak{R} = \langle X^\mathfrak{R}, Y^\mathfrak{R}, I^\mathfrak{R}\rangle$ as defined in Definition 3.13 is an R-context with objects taken from C, attributes being derived from attributes in $Y^\sharp$ which are not present in D. Note

that for each $\langle n, B\rangle \in Y^\sharp$ which is not in D, $Y^\mathfrak{R}$ contains an attribute $\langle n', B\rangle \in Y^\mathfrak{R}$, where n' is a new flag. The value of the flag is either the same if $f(\langle n, B\rangle)$ is greater or equal to $f(\min(D))$ or the flag is equal to the size of B. In other words, if $\langle n, B\rangle$ is behind $\min(D)$ in terms of the order of attributes, the flag is not updated. The most important part of the flag update is that an attribute $\langle 0, B\rangle \in Y^\sharp$ will be given a nonzero flag in $Y^\mathfrak{R}$ if it is not in D and if it stays before $\min(D)$ in terms of the order of attributes.

In general, $\mathbb{K}^\mathfrak{R} = \langle X^\mathfrak{R}, Y^\mathfrak{R}, I^\mathfrak{R}\rangle$ can contain two or more indistinguishable attributes (equal columns in the corresponding data table), i.e., $\mathbb{K}^\mathfrak{R}$ may not be clarified in sense of Definition 3.6. The algorithm described in the next section relies on reduction and clarification of R-contexts, we therefore introduce the following notation:

Definition 3.15. If $\mathbb{K}^\mathfrak{R}$ results from $\mathbb{K}^\sharp$ using C and D in sense of Definition 3.13 and if $\mathbb{K}^\mathfrak{C}$ is a clarification of $\mathbb{K}^\mathfrak{R}$ in sense of Definition 3.6, then $\mathbb{K}^\mathfrak{C}$ will be denoted by $\textsc{Reduce}(\mathbb{K}^\sharp, C, D)$. ■

Table 3. Context from Definition 3.13 (left), result of REDUCE (middle), and its concise representation (right).

$\mathbb{K}^\mathfrak{R}$	$\langle 1,\{6\}\rangle$	$\langle 0,\{3\}\rangle$	$\langle 1,\{2,7\}\rangle$
d	×		×
e			

$\mathbb{K}^\mathfrak{R}$	$\langle 0,\{3\}\rangle$	$\langle 2,\{2,6,7\}\rangle$
d		×
e		

$\mathbb{K}^\mathfrak{R}$	$\{3\}$	$\{2,6,7\}$
d		×
e		

Example 3.16. Consider R-context from Table 2 (right). For $C = \{d, e\}$, and $D = \{\langle 0, \{1, 4\}\rangle\}$, the R-context $\mathbb{K}^\mathfrak{R}$ specified in Definition 3.13 is depicted in Table 3 (left). Notice that during the reduction, attribute $\langle 1, \{6\}\rangle$ was given a nonzero flag since its position according to f was before that of attribute $\langle 0, \{1, 4\}\rangle$ in the original R-context. Table 3 (middle) represents a clarified version of $\mathbb{K}^\mathfrak{R}$ with attributes sorted according to their supports. Hence, the middle table represents the result of REDUCE. Table 3 (right) is a concise representation of the same R-context in which columns corresponding to attributes with nonzero flags are highlighted as gray (the descriptions of attributes then contain just sets of original attributes, the numerical flags are omitted).

4. Algorithm

In this section, we describe the proposed algorithm for computing formal concepts. The main part of the algorithm is a recursive procedure COMPUTE from Algorithm 1. The procedure accepts as its argument a clarified R-context and during the computation it calls an auxiliary procedure CLOSURE from Algorithm 2.

When invoked with $\mathbb{K}^\sharp$, procedure COMPUTE proceeds as follows. First, it stores a tuple which consists of the set of objects $X^\sharp$ and $\textsc{Int}(\mathbb{K}^\sharp, Y)$, where

$$\textsc{Int}(\mathbb{K}^\sharp, Y) = Y \setminus \lfloor Y^\sharp \rfloor. \tag{6}$$

Recall that $\lfloor \cdots \rfloor$ is defined by (1). Thus, $\textsc{Int}(\mathbb{K}^\sharp, Y) = Y \setminus \bigcup\{B \subseteq Y \mid \langle n, B\rangle \in Y^\sharp\}$. Then, the procedure goes over all attributes in $Y^\sharp$ with zero flags (see lines 2 and 3 of Algorithm 1). For each such

Algorithm 1: Procedure COMPUTE$(\mathbb{K}^\sharp)$

```
1  store ⟨X♯, INT(K♯, Y)⟩;
2  for ⟨n, B⟩ ∈ Y♯ do
3      if n = 0 then
4          set ⟨C, D⟩ to CLOSURE(K♯, ⟨n, B⟩);
5          if Σ{n ∈ ℕ0 | ⟨n, B⟩ ∈ D} = 0 then
6              COMPUTE(REDUCE(K♯, C, D));
7          end
8      end
9  end
10 return
```

Algorithm 2: Procedure CLOSURE$(\mathbb{K}^\sharp, \langle n, B\rangle)$

```
1  C = {x ∈ X♯ | ⟨x, ⟨n, B⟩⟩ ∈ I♯};
2  D = {y ∈ Y♯ | f(⟨n, B⟩) ≤ f(y)};
3  for x ∈ C do
4      for y ∈ D do
5          if ⟨x, y⟩ ∉ I♯ then
6              remove y from D;
7          end
8      end
9  end
10 return ⟨C, D⟩
```

attribute $\langle n, B\rangle$, the procedure invokes CLOSURE and the result of invocation is stored in $\langle C, D\rangle$. An easy inspection of the pseudocode in Algorithm 2 shows that the result of calling CLOSURE$(\mathbb{K}^\sharp, \langle n, B\rangle)$ is a formal concept in $\mathbb{K}^\sharp$ generated by attribute $\langle n, B\rangle$, i.e., $C = \{\langle n, B\rangle\}^{\downarrow_{\mathbb{K}^\sharp}}$ and $D = C^{\uparrow_{\mathbb{K}^\sharp}}$. Notice that Algorithm 2 utilizes attribute sorting together with the fact that $\mathbb{K}^\sharp$ is clarified. In that case, all attributes which belong to D must have their indices strictly greater than or equal to $f(\langle n, B\rangle)$. This observation has already been made in [13].

Next step of Algorithm 1 is a canonicity test which succeeds iff all flags in D (computed in the previous step) are zero, see line 5. In the case of success, COMPUTE invokes itself with reduced (and clarified) formal context which results from $\mathbb{K}^\sharp$, see line 6. Otherwise, the algorithm continues with another attribute. When all attributes are processed, the invocation of COMPUTE for $\mathbb{K}^\sharp$ is left.

For input formal context $\mathbb{K} = \langle X, Y, I\rangle$, the first invocation of COMPUTE can be described by the following consecutive steps:

1. take an initial R-context $\mathbb{K}^\sharp = \langle X^\sharp, Y^\sharp, I^\sharp\rangle$ derived from $\mathbb{K}$;
2. determine a clarified R-context $\mathbb{K}^\complement = \langle X^\complement, Y^\complement, I^\complement\rangle$ which results from $\mathbb{K}^\sharp$;
3. if $|\{\langle n, B\rangle\}^{\downarrow_{\mathbb{K}^\complement}}| < |X^\complement|$ for all $\langle n, B\rangle \in Y^\complement$ then call COMPUTE$(\mathbb{K}^\complement)$.

4. if there is $\langle n, B\rangle \in Y^{\complement}$ such that $|\{\langle n, B\rangle\}^{\downarrow_{\mathbb{K}^{\complement}}}| = |X^{\complement}|$, then call COMPUTE$(\mathbb{K}^*)$, where $\mathbb{K}^* = \langle X^*, Y^*, I^*\rangle$ with $X^* = X^{\complement}$, $Y^* = Y^{\complement} \setminus \{\langle n, B\rangle\}$, and $I^* = I^{\complement} \cap (X^* \times Y^*)$.

In other words, $\mathbb{K}$ is transformed into an R-context and clarified. If the resulting R-context contains an attribute shared by all objects (notice that since it is clarified, such an attribute is at most one), it is removed from the R-context. Then, COMPUTE is invoked with such R-context as an input. In Section 6, we shall prove that the algorithm is sound, i.e., with input data of this form, it stores all formal concepts, each of them exactly once.

Remark 4.1. Notice that the canonicity test is expressed using a sum, see line 5 of Algorithm 1. One can easily see that we might as well use "logical or" provided that all flags are assigned values 0 and 1, only. This can be achieved by slight modifications of $\mathrm{Attr}(\langle n, B\rangle)$ which appears in Definition 3.13 and $Y^{\complement}$ defined in Definition 3.6. Indeed, the numerical value of the flag is not as important for the algorithm. The important fact is whether at least one of the attributes in intent D has nonzero flag, see Algorithm 1.

Table 4. Illustrative formal context

$\mathbb{K}$	0	1	2	3	4	5
a		×	×	×		
b	×	×				×
c		×		×	×	
d	×		×		×	

5. Illustrative Example

Before we investigate properties of Algorithm 1, we show here an illustrative running example in which we demonstrate how COMPUTE behaves for particular input data. This illustration is useful for getting first (informal) insight into the algorithm. Consider an input formal context $\mathbb{K} = \langle X, Y, I\rangle$ with objects $X = \{a, b, c, d\}$, attributes $Y = \{0, 1, 2, 3, 4, 5\}$, and $I \subseteq X \times Y$ as in Table 4. One can check that $\mathbb{K}$ has 11 formal concepts, namely:

$$
\begin{aligned}
R_1 &= \langle\{a, b, c, d\}, \emptyset\rangle, & R_5 &= \langle\{d\}, \{0, 2, 4\}\rangle, & R_9 &= \langle\{c\}, \{1, 3, 4\}\rangle,\\
R_2 &= \langle\{b\}, \{0, 1, 5\}\rangle, & R_6 &= \langle\{a, d\}, \{2\}\rangle, & R_{10} &= \langle\{c, d\}, \{4\}\rangle,\\
R_3 &= \langle\emptyset, \{0, 1, 2, 3, 4, 5\}\rangle, & R_7 &= \langle\{a\}, \{1, 2, 3\}\rangle, & R_{11} &= \langle\{a, b, c\}, \{1\}\rangle.\\
R_4 &= \langle\{b, d\}, \{0\}\rangle, & R_8 &= \langle\{a, c\}, \{1, 3\}\rangle,
\end{aligned}
$$

Algorithm 1 proceeds for $\mathbb{K}$ as follows. First, an initial and clarified R-context is created, denote it by $\mathbb{K}_1^\sharp$. Since in $\mathbb{K}$ all attributes are distinct and there is no attibute which is shared by all objects, $\mathbb{K}_1^\sharp$ is directly passed to COMPUTE as the initial argument. The initial R-context is depicted in Figure 1 (top).

The execution of COMPUTE proceeds with selecting an attribute from $Y_1^\sharp$, computing the closure and reduction $\mathbb{K}_2^\sharp$, and recursive invocation of COMPUTE:

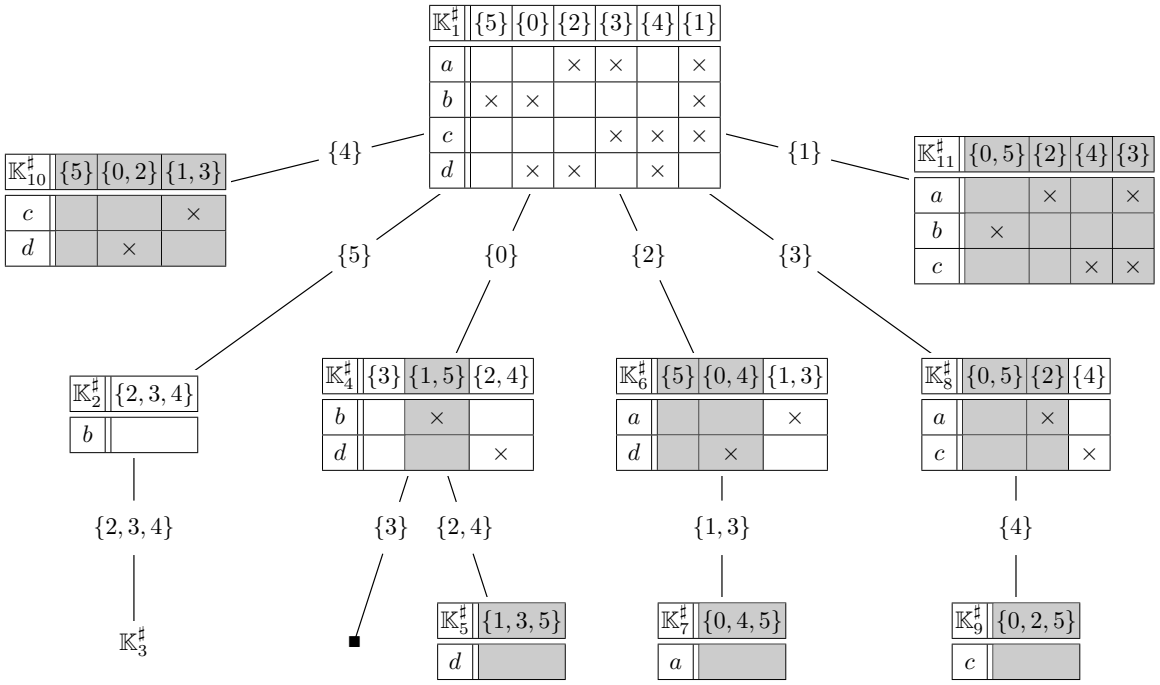

Figure 1. R-contexts produced by Algorithm 1 during computation.

line 1: **store** $\langle X_1^\sharp, \textsc{Int}(\mathbb{K}_1^\sharp, Y)\rangle = \langle \boldsymbol{\{a,b,c,d\}, \emptyset}\rangle = \boldsymbol{R_1}$
line 4: set $\langle C_2, D_2\rangle$ to $\textsc{Closure}(\mathbb{K}_1^\sharp, \langle 0, \{5\}\rangle) = \langle \{b\}, \{\langle 0, \{5\}\rangle, \langle 0, \{0\}\rangle, \langle 0, \{1\}\rangle\}\rangle$
line 5: success for $\langle C_2, D_2\rangle = \langle \{b\}, \{\langle 0, \{5\}\rangle, \langle 0, \{0\}\rangle, \langle 0, \{1\}\rangle\}\rangle$ because all flags are 0
line 6: call $\textsc{Compute}(\mathbb{K}_2^\sharp)$ for $\mathbb{K}_2^\sharp = \textsc{Reduce}(\mathbb{K}_1^\sharp, C_2, D_2)$

Notice that in Figure 1, the recursive invocation is depicted by an R-context $\mathbb{K}_2^\sharp$ connected with $\mathbb{K}_1^\sharp$ with an edge labeled by $\{5\}$ which is the original set of attributes present in attribute $\langle 0, \{5\}\rangle \in Y_1^\sharp$. Moreover, the computation continues as follows:

line 1: **store** $\langle X_2^\sharp, \textsc{Int}(\mathbb{K}_2^\sharp, Y)\rangle = \langle \boldsymbol{\{b\}, \{0,1,5\}}\rangle = \boldsymbol{R_2}$
line 4: set $\langle C_3, D_3\rangle$ to $\textsc{Closure}(\mathbb{K}_2^\sharp, \langle 0, \{2,3,4\}\rangle) = \langle \emptyset, \{\langle 0, \{2,3,4\}\rangle\}\rangle$
line 5: success for $\langle C_3, D_3\rangle = \langle \emptyset, \{\langle 0, \{2,3,4\}\rangle\}\rangle$ because all flags are 0
line 6: call $\textsc{Compute}(\mathbb{K}_3^\sharp)$ for $\mathbb{K}_3^\sharp = \textsc{Reduce}(\mathbb{K}_2^\sharp, C_3, D_3)$
  line 1: **store** $\langle X_3^\sharp, \textsc{Int}(\mathbb{K}_3^\sharp, Y)\rangle = \langle \boldsymbol{\emptyset, \{0,1,2,3,4,5\}}\rangle = \boldsymbol{R_3}$
  $\perp$ return from invocation of $\textsc{Compute}$ for $\mathbb{K}_3^\sharp$
$\perp$ return from invocation of $\textsc{Compute}$ for $\mathbb{K}_2^\sharp$

Notice that since $\mathbb{K}_3^\sharp$ is a trivial context with empty sets of objects and attributes, the invocation of $\textsc{Compute}$ has immediatelly returned after storing R_3 because the iteration of the for-loop is trivially done for empty $Y_3^\sharp$. Next, the computation resumes in the first invocation of $\textsc{Compute}$ considering next attribute:

line 4: set $\langle C_4, D_4 \rangle$ to CLOSURE($\mathbb{K}_1^\sharp, \langle 0, \{0\} \rangle$) = $\langle \{b, d\}, \{\langle 0, \{0\} \rangle\} \rangle$
line 5: success for $\langle C_4, D_4 \rangle = \langle \{b, d\}, \{\langle 0, \{0\} \rangle\} \rangle$ because all flags are 0
line 6: call COMPUTE($\mathbb{K}_4^\sharp$) for $\mathbb{K}_4^\sharp$ = REDUCE($\mathbb{K}_1^\sharp, C_4, D_4$)
  line 1: **store** $\langle X_4^\sharp, \text{INT}(\mathbb{K}_4^\sharp, Y) \rangle = \boldsymbol{\langle \{b, d\}, \{0\} \rangle = R_4}$
  line 4: set $\langle C_5, D_5 \rangle$ to CLOSURE($\mathbb{K}_4^\sharp, \langle 0, \{3\} \rangle$) = $\langle \emptyset, \{\langle 0, \{3\} \rangle, \langle 1, \{1, 5\} \rangle, \langle 0, \{2, 4\} \rangle\} \rangle$
  line 5: failure for $\langle C_5, D_5 \rangle = \langle \emptyset, \{\langle 0, \{3\} \rangle, \langle 1, \{1, 5\} \rangle, \langle 0, \{2, 4\} \rangle\} \rangle$ because $\langle 1, \{1, 5\} \rangle \in D_5$

At this point, the canonicity test has failed. Therefore, the algorithm does not continue with $\langle C_5, D_5 \rangle$ which in fact determines formal concept R_3 that has been computed and processed before. This is the only point where the canonicity test fails in this example and where a concept is computed more than once. Notice that it is not the case that R_3 is computed as such, the algorithm has computed $\langle C_5, D_5 \rangle$ but anyhow, $\langle C_5, D_5 \rangle$ would normally be used to determine R_3, i.e. we can consider R_3 to be computed twice. In Figure 1 the situation is depicted by a black square node labeled by R_3. After this point, the computation continues as follows (without further comments):

  line 4: set $\langle C_5, D_5 \rangle$ to CLOSURE($\mathbb{K}_4^\sharp, \langle 0, \{2, 4\} \rangle$) = $\langle \{d\}, \{\langle 0, \{2, 4\} \rangle\} \rangle$
  line 5: success for $\langle C_5, D_5 \rangle = \langle \{d\}, \{\langle 0, \{2, 4\} \rangle\} \rangle$ because all flags are 0
  line 6: call COMPUTE($\mathbb{K}_5^\sharp$) for $\mathbb{K}_5^\sharp$ = REDUCE($\mathbb{K}_4^\sharp, C_5, D_5$)
    line 1: **store** $\langle X_5^\sharp, \text{INT}(\mathbb{K}_5^\sharp, Y) \rangle = \boldsymbol{\langle \{d\}, \{0, 2, 4\} \rangle = R_5}$
    $\perp$ return from invocation of COMPUTE for $\mathbb{K}_5^\sharp$
  $\perp$ return from invocation of COMPUTE for $\mathbb{K}_4^\sharp$
line 4: set $\langle C_6, D_6 \rangle$ to CLOSURE($\mathbb{K}_1^\sharp, \langle 0, \{2\} \rangle$) = $\langle \{a, d\}, \{\langle 0, \{2\} \rangle\} \rangle$
line 5: success for $\langle C_6, D_6 \rangle = \langle \{a, d\}, \{\langle 0, \{2\} \rangle\} \rangle$ because all flags are 0
line 6: call COMPUTE($\mathbb{K}_6^\sharp$) for $\mathbb{K}_6^\sharp$ = REDUCE($\mathbb{K}_1^\sharp, C_6, D_6$)
  line 1: **store** $\langle X_6^\sharp, \text{INT}(\mathbb{K}_6^\sharp, Y) \rangle = \boldsymbol{\langle \{a, d\}, \{2\} \rangle = R_6}$
  line 4: set $\langle C_7, D_7 \rangle$ to CLOSURE($\mathbb{K}_6^\sharp, \langle 0, \{1, 3\} \rangle$) = $\langle \{a\}, \{\langle 0, \{1, 3\} \rangle\} \rangle$
  line 5: success for $\langle C_7, D_7 \rangle = \langle \{a\}, \{\langle 0, \{1, 3\} \rangle\} \rangle$ because all flags are 0
  line 6: call COMPUTE($\mathbb{K}_7^\sharp$) for $\mathbb{K}_7^\sharp$ = REDUCE($\mathbb{K}_6^\sharp, C_7, D_7$)
    line 1: **store** $\langle X_7^\sharp, \text{INT}(\mathbb{K}_7^\sharp, Y) \rangle = \boldsymbol{\langle \{a\}, \{1, 2, 3\} \rangle = R_7}$
    $\perp$ return from invocation of COMPUTE for $\mathbb{K}_7^\sharp$
  $\perp$ return from invocation of COMPUTE for $\mathbb{K}_6^\sharp$
line 4: set $\langle C_8, D_8 \rangle$ to CLOSURE($\mathbb{K}_1^\sharp, \langle 0, \{3\} \rangle$) = $\langle \{a, c\}, \{\langle 0, \{3\} \rangle, \langle 0, \{1\} \rangle\} \rangle$
line 5: success for $\langle C_8, D_8 \rangle = \langle \{a, c\}, \{\langle 0, \{3\} \rangle, \langle 0, \{1\} \rangle\} \rangle$ because all flags are 0
line 6: call COMPUTE($\mathbb{K}_8^\sharp$) for $\mathbb{K}_8^\sharp$ = REDUCE($\mathbb{K}_1^\sharp, C_8, D_8$)
  line 1: **store** $\langle X_8^\sharp, \text{INT}(\mathbb{K}_8^\sharp, Y) \rangle = \boldsymbol{\langle \{a, c\}, \{1, 3\} \rangle = R_8}$
  line 4: set $\langle C_9, D_9 \rangle$ to CLOSURE($\mathbb{K}_8^\sharp, \langle 0, \{4\} \rangle$) = $\langle \{c\}, \{\langle 0, \{4\} \rangle\} \rangle$
  line 5: success for $\langle C_9, D_9 \rangle = \langle \{c\}, \{\langle 0, \{4\} \rangle\} \rangle$ because all flags are 0
  line 6: call COMPUTE($\mathbb{K}_9^\sharp$) for $\mathbb{K}_9^\sharp$ = REDUCE($\mathbb{K}_8^\sharp, C_9, D_9$)
    line 1: **store** $\langle X_9^\sharp, \text{INT}(\mathbb{K}_9^\sharp, Y) \rangle = \boldsymbol{\langle \{c\}, \{1, 3, 4\} \rangle = R_9}$
    $\perp$ return from invocation of COMPUTE for $\mathbb{K}_9^\sharp$
  $\perp$ return from invocation of COMPUTE for $\mathbb{K}_8^\sharp$
line 4: set $\langle C_{10}, D_{10} \rangle$ to CLOSURE($\mathbb{K}_1^\sharp, \langle 0, \{4\} \rangle$) = $\langle \{c, d\}, \{\langle 0, \{4\} \rangle\} \rangle$

line 5: success for $\langle C_{10}, D_{10}\rangle = \langle\{c, d\}, \{\langle 0, \{4\}\rangle\}\rangle$ because all flags are 0
line 6: call COMPUTE($\mathbb{K}^\sharp_{10}$) for $\mathbb{K}^\sharp_{10} = $ REDUCE($\mathbb{K}^\sharp_1, C_{10}, D_{10}$)
  line 1: **store** $\langle X^\sharp_{10}, \text{INT}(\mathbb{K}^\sharp_{10}, Y)\rangle = \boldsymbol{\langle\{c, d\}, \{4\}\rangle = R_{10}}$
  $\perp$ return from invocation of COMPUTE for $\mathbb{K}^\sharp_{10}$
line 4: set $\langle C_{11}, D_{11}\rangle$ to CLOSURE($\mathbb{K}^\sharp_1, \langle 0, \{1\}\rangle$) = $\langle\{a, b, c\}, \{\langle 0, \{1\}\rangle\}\rangle$
line 5: success for $\langle C_{11}, D_{11}\rangle = \langle\{a, b, c\}, \{\langle 0, \{1\}\rangle\}\rangle$ because all flags are 0
line 6: call COMPUTE($\mathbb{K}^\sharp_{11}$) for $\mathbb{K}^\sharp_{11} = $ REDUCE($\mathbb{K}^\sharp_1, C_{11}, D_{11}$)
  line 1: **store** $\langle X^\sharp_{11}, \text{INT}(\mathbb{K}^\sharp_{11}, Y)\rangle = \boldsymbol{\langle\{a, b, c\}, \{1\}\rangle = R_{11}}$
  $\perp$ return from invocation of COMPUTE for $\mathbb{K}^\sharp_{11}$
$\perp$ return from invocation of COMPUTE for $\mathbb{K}^\sharp_1$

Remark 5.1. It is interesting to compare the presented algorithm with CbO [14, 15, 16] and FCbO [13, 23] in terms of formal concepts which are computed multiple times. In a similar way as in the case of our algorithm, CbO an FCbO are recursively invoked and the computation can therefore be expressed by a corresponding call tree. Figure 2 shows a call tree for both CbO and FCbO applied to input formal context from the example. The bold lines correspond to both CbO and FCbO, the dotted lines correspond only to CbO. The black square nodes labeled by formal concepts represent branches of computation where the concepts are computed but fail the canonicity test. We can see that FCbO computes 7 formal concepts which fail the canonicity test. Thus, several concepts are computed multiple times. Namely, R_2 is computed twice, R_3 is computed three times, so is R_5, and R_8 and R_9 and both computed twice. Recall that our algorithm computes just a single formal concept twice, so this is an interesting improvement. In the case of CbO, the improvement is even more visible since here the number of computed concepts which fail the canonicity test is 19. Section 7 shows experimental evaluation of average behavior of our algorithm compared to CbO and FCbO using various data sets which shows an interesting tendency that the numbers of formal concepts computed multiple times by the presented algorithm are much smaller.

6. Algorithm Properties and Soundness

In this section, we pay attention to properties of the algorithm and prove its soundness which means that for an input formal context, the algorithm stores each formal concept exactly once. In other words, if a formal concept is calculated several times, the algorithm ensures that it is stored (e.g., printed as an output or stored in an output data structure) at most once; moreover, the algorithm ensures that each formal concept is stored at least once. The two conditions together yield that each formal concept is stored exactly once.

We take the same assumptions as in Section 4. Hence, we assume that $\mathbb{K} = \langle X, Y, I\rangle$ is the input formal context and that COMPUTE is invoked according to the steps described in Section 4.

In order to prove soundness of the algorithm, we first show that each R-context which is passed to COMPUTE as an argument during the computation represents a formal context. That is, if one considers line 1 of Algorithm 1, for such an R-context, the algorithm stores a couple which is a formal concept in $\mathbb{K}$. Notice that one can easily find an R-context for which this is not so. Therefore, we introduce the following notion.

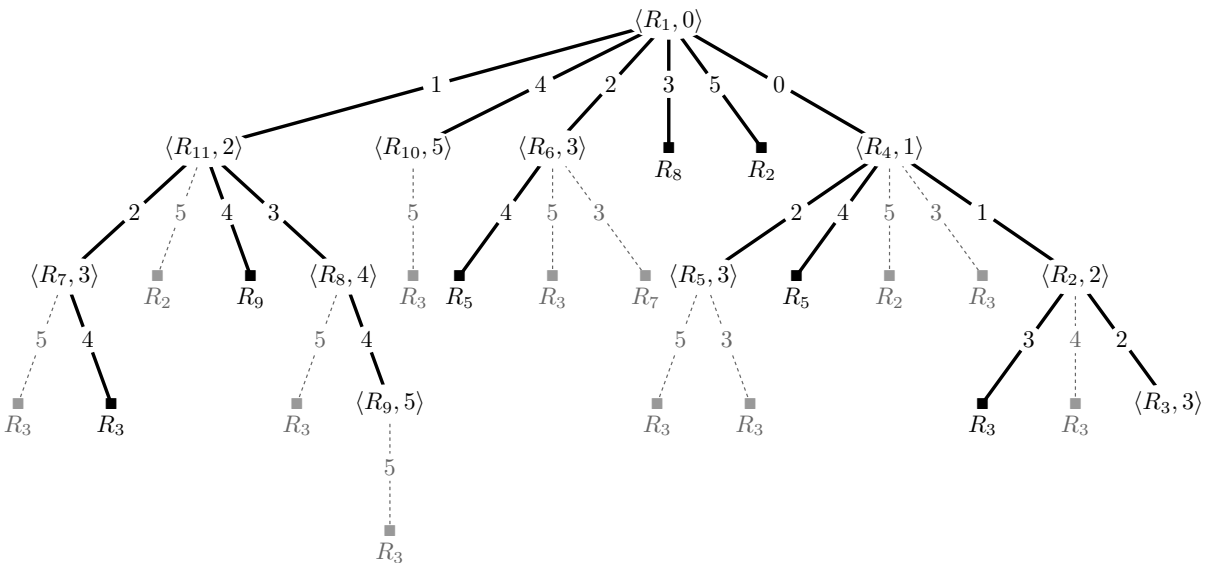

Figure 2. Example of a call tree of FCbO with reduced number of leaf nodes.

Definition 6.1. Let $\mathbb{K}^\sharp$ be an R-context derived from $\mathbb{K}$. We shall say that $\mathbb{K}^\sharp$ is $\mathbb{K}$-representative if $\langle X^\sharp, \textsc{Int}(\mathbb{K}^\sharp, Y)\rangle$ is a formal concept in $\mathbb{K}$. ■

The definition captures exactly the property that is needed to store (only) formal concepts. The next assertion shows that the property is preserved during consecutive invocations of COMPUTE.

Lemma 6.2. Let $\mathbb{K}^\sharp$ be a $\mathbb{K}$-representative R-context derived from $\mathbb{K}$ and let $\langle C, D\rangle$ be a formal concept in $\mathbb{K}^\sharp$ with $D \neq \emptyset$. Then, $\textsc{Reduce}(\mathbb{K}^\sharp, C, D)$ is $\mathbb{K}$-representative.

Proof:
Denote $\textsc{Reduce}(\mathbb{K}^\sharp, C, D)$ by $\mathbb{K}^\mathfrak{R}$. Since $\mathbb{K}^\sharp$ is $\mathbb{K}$-representative, we have that $(X^\sharp)^{\uparrow_\mathbb{K}} = \textsc{Int}(\mathbb{K}^\sharp, Y)$ and $\textsc{Int}(\mathbb{K}^\sharp, Y)^{\downarrow_\mathbb{K}} = X^\sharp$. Moreover, since $\langle C, D\rangle$ is assumed to be a formal concept in $\mathbb{K}^\sharp$, we have $C^{\uparrow_{\mathbb{K}^\sharp}} = D$ and $D^{\downarrow_{\mathbb{K}^\sharp}} = C$. We now show that $\langle C, \textsc{Int}(\mathbb{K}^\sharp, Y) \cup \lfloor D \rfloor\rangle$ is a formal concept in $\mathbb{K}$. Notice that according to Definition 3.13, this would prove that $\mathbb{K}^\mathfrak{R}$ is $\mathbb{K}$-representative because by Definition 3.13, we have $\textsc{Int}(\mathbb{K}^\mathfrak{R}, Y) = \textsc{Int}(\mathbb{K}^\sharp, Y) \cup \lfloor D \rfloor$.

Using (2), we get $\lfloor D \rfloor = \lfloor C^{\uparrow_{\mathbb{K}^\sharp}} \rfloor \subseteq C^{\uparrow_\mathbb{K}}$. Since $C \subseteq X^\sharp$, we get $\textsc{Int}(\mathbb{K}^\sharp, Y) = (X^\sharp)^{\uparrow_\mathbb{K}} \subseteq C^{\uparrow_\mathbb{K}}$. Putting the inclusions together, we get $\textsc{Int}(\mathbb{K}^\sharp, Y) \cup \lfloor D \rfloor \subseteq C^{\uparrow_\mathbb{K}}$. In order to prove the converse inclusion, it suffices to check that if $y \in C^{\uparrow_\mathbb{K}}$ and $y \notin \textsc{Int}(\mathbb{K}^\sharp, Y)$, then $y \in \lfloor D \rfloor$. If $y \notin \textsc{Int}(\mathbb{K}^\sharp, Y)$, there is $\langle n, B\rangle \in Y^\sharp$ such that $y \in B$. If in addition $y \in C^{\uparrow_\mathbb{K}}$, then $\langle x, y\rangle \in I$ for all $x \in C$. Using Definition 3.1 (iii), $\langle x, y\rangle \in I$ for all $x \in C$ and all $y \in B$, meaning that $\langle x, \langle n, B\rangle\rangle \in I^\sharp$ for all $x \in C$. Hence, $\langle n, B\rangle \in C^{\uparrow_{\mathbb{K}^\sharp}} = D$ which yields $y \in B \subseteq \lfloor D \rfloor$. Altogether, $C^{\uparrow_\mathbb{K}} = \textsc{Int}(\mathbb{K}^\sharp, Y) \cup \lfloor D \rfloor$. Now, using (3), we get $(\textsc{Int}(\mathbb{K}^\sharp, Y) \cup \lfloor D \rfloor)^{\downarrow_\mathbb{K}} = \textsc{Int}(\mathbb{K}^\sharp, Y)^{\downarrow_\mathbb{K}} \cap \lfloor D \rfloor^{\downarrow_\mathbb{K}} = X^\sharp \cap \lfloor D \rfloor^{\downarrow_\mathbb{K}} = D^{\downarrow_{\mathbb{K}^\sharp}} = C$. □

Corollary 6.3. All tuples stored during invocations of COMPUTE are formal concepts in $\mathbb{K}$.

Proof:
The proof is obvious. Indeed, by induction and using Lemma 6.2, one can check that each R-context that is passed to COMPUTE is $\mathbb{K}$-representative. □

Notice that now it becomes apparent why we have removed an attribute shared by all objects from the clarified initial R-context (see step 4 described in Section 4). Otherwise, the argument for the first invocation of COMPUTE would not be $\mathbb{K}$-representative, meaning that COMPUTE would store a pair which is not a formal concept (the attribute shared by all objects would not be present in intent of the first stored pair).

The following assertion shows that Algorithm 1 provides a complete search for formal concepts, i.e., each formal concept is stored at least once.

Lemma 6.4. During the invocations of COMPUTE, each formal concept in $\mathbb{K}$ is stored at least once.

Proof:
For brevity, we denote by $\mathbb{K}_i^\sharp \prec \mathbb{K}_j^\sharp$ the fact that if COMPUTE is invoked with $\mathbb{K}_i^\sharp$, then during its invocation, it invokes itself with $\mathbb{K}_j^\sharp$. Therefore, $\mathbb{K}_j^\sharp$ is equal to $\text{REDUCE}(\mathbb{K}_i^\sharp, C, D)$ for some C and D.

Take formal concept $\langle E, F\rangle$ in $\mathbb{K}$. We prove that there is a sequence $\mathbb{K}_1^\sharp \prec \cdots \prec \mathbb{K}_n^\sharp$ of $\mathbb{K}$-representative R-contexts derived from $\mathbb{K}$ such that $X_n^\sharp = E$ and $\mathbb{K}_1^\sharp$ is the argument of the first invocation of COMPUTE. This would prove that formal concept $\langle E, F\rangle$ will be stored by COMPUTE invoked with $\mathbb{K}_n^\sharp$.

We construct the sequence as follows. The first element $\mathbb{K}_1^\sharp$ is determined uniquely. Assume that we have constructed first i elements of the sequence and for $\mathbb{K}_i^\sharp$ we have that if $\langle n, B\rangle \in Y_i^\sharp$ and $B \subseteq F$, then $n = 0$. Observe that this property holds for $\mathbb{K}_1^\sharp$ trivially since the flags of all attributes in $Y_1^\sharp$ are all zero. If $X_i^\sharp = E$, we are done. Otherwise, we show that we can choose a $\mathbb{K}$-representative R-context $\mathbb{K}_{i+1}^\sharp$ derived from $\mathbb{K}$ such that $\mathbb{K}_i^\sharp \prec \mathbb{K}_{i+1}^\sharp$ for which we have that if $\langle n, B\rangle \in Y_{i+1}^\sharp$ and $B \subseteq F$, then $n = 0$. Thus, if $X_i^\sharp \supset E$, then $\lfloor Y_i^\sharp \rfloor \cap F \neq \emptyset$ because $\mathbb{K}_i^\sharp$ is $\mathbb{K}$-representative. Thus, we can take $\langle n, B\rangle \in Y_i^\sharp$ such that $B \cap F \neq \emptyset$ and $f(\langle n, B\rangle) \leq f(\langle n', B'\rangle)$ is true for all $\langle n', B'\rangle \in Y_i^\sharp$ satisfying $B' \cap F \neq \emptyset$. Recall that f is the bijective map which determines the order (i.e., the indices) of attributes in $\mathbb{K}_i^\sharp$.

Moreover, $B \cap F \neq \emptyset$ yields there is $y \in B$ such that $y \in E^{\uparrow_{\mathbb{K}}}$. Hence, $y \in \lfloor E^{\uparrow_{\mathbb{K}_i^\sharp}} \rfloor$ which in turn means that $B \subseteq \lfloor E^{\uparrow_{\mathbb{K}_i^\sharp}} \rfloor$ because all attributes from B are indistinguishable on objects from $E \subseteq X_i^\sharp$. Therefore, $B \subseteq \lfloor E^{\uparrow_{\mathbb{K}_i^\sharp}} \rfloor \subseteq F$. Since $B \subseteq F$ and $\langle n, B\rangle \in Y_i^\sharp$, we have by assumption $n = 0$. Hence, for attribute $\langle n, B\rangle$, Algoritm 1 can proceed to line 4. Let $\langle C, D\rangle$ be defined by $C = \{\langle n, B\rangle\}^{\downarrow_{\mathbb{K}_i^\sharp}}$ and $D = C^{\uparrow_{\mathbb{K}_i^\sharp}}$ which corresponds to calling CLOSURE with $\mathbb{K}_i^\sharp$ and $\langle n, B\rangle$ as its arguments. We now check that the canonicity test succeeds. If $\langle n', B'\rangle \in D$, then clearly $B' \subseteq F$ because $B \subseteq F$ and $\langle x, \langle n', B'\rangle\rangle \in I_i^\sharp$ holds for any $x \in X_i^\sharp$ such that $\langle x, \langle n, B\rangle\rangle \in I_i^\sharp$. Hence, using the assumption, it follows that $n' = 0$. Thus, all flags of attributes from D are zero, i.e. the canonicity test succeeds. As a consequence, we can put $\mathbb{K}_{i+1}^\sharp = \text{REDUCE}(\mathbb{K}_i^\sharp, C, D)$ and we have $\mathbb{K}_i^\sharp \prec \mathbb{K}_{i+1}^\sharp$. Moreover, Lemma 6.2 yields that $\mathbb{K}_{i+1}^\sharp$ is $\mathbb{K}$-representative.

It remains to show that $\mathbb{K}^\sharp_{i+1}$ satisfies the property that if $\langle n, B\rangle \in Y^\sharp_{i+1}$ and $B \subseteq F$, then $n = 0$. Take any $\langle n, B\rangle \in Y^\sharp_{i+1}$ such that $B \subseteq F$. Since $\mathbb{K}^\sharp_{i+1}$ results by reduction and clarification, there are $\langle n_j, B_j\rangle \in Y^\sharp_i$ ($j \in J$) such that $B = \bigcup_{j\in J} B_j$. We have $B_j \subseteq F$ ($j \in J$), i.e., $n_j = 0$. Since $\mathbb{K}^\sharp_{i+1} = \textsc{Reduce}(\mathbb{K}^\sharp_i, C, D)$ for $C = \{\langle n, B\rangle\}^{\downarrow_{\mathbb{K}^\sharp_i}}$ where $\langle n, B\rangle$ was chosen with the least possible index according to f, during the reduction, no attribute $\langle n_j, B_j\rangle$ was given a nonzero flag. During the subsequent clarification, some of the attributes $\langle n_j, B_j\rangle$ can be merged together with other attributes with zero flags but they cannot be merged with attributes with nonzero flags (otherwise, it would contradict the fact that $\langle E, F\rangle$ is a formal concept). Therefore, $n = \sum_{j\in J} n_j = 0$, proving the property for $\mathbb{K}^\sharp_{i+1}$.

In order to finish the proof, observe that the sequence can be extended only finitely may times and for $\mathbb{K}^\sharp_i \prec \mathbb{K}^\sharp_{i+1}$, we have $X^\sharp_i \supset X^\sharp_{i+1} \supseteq E$. Hence, after finitely many steps, we obtain $\mathbb{K}^\sharp_n$ with $E = X^\sharp_n$ and thus $F = \textsc{Int}(\mathbb{K}^\sharp_n, Y)$ since $\mathbb{K}^\sharp_n$ if $\mathbb{K}$-representative. □

Theorem 6.5. (soundness of Algorithm 1)
During the invocations of COMPUTE, each formal concept in $\mathbb{K}$ is stored exactly once.

Proof:
Using Lemma 6.4, each formal concept in $\mathbb{K}$ is stored at least once. Thus, it suffices to prove that each of them is stored at most once. We prove this by showing uniqueness of sequences constructed in the proof of Lemma 6.4. Inspecting the proof of Lemma 6.4, one can see that $\mathbb{K}^\sharp_{i+1}$ is determined from $\mathbb{K}^\sharp_i$ by reduction which uses a formal concept in $\mathbb{K}^\sharp_i$ generated by the least possible attribute $\langle n, B\rangle \in Y^\sharp_i$ such that $B \subseteq F$. If we would have chosen other attribute $\langle n', B'\rangle \in Y^\sharp_i$ such that $B' \subseteq F$ instead of $\langle n, B\rangle$, then $\mathbb{K}^\sharp_{i+1} = \textsc{Reduce}(\mathbb{K}^\sharp_i, C, D)$ for $C = \{\langle n', B'\rangle\}^{\downarrow_{\mathbb{K}^\sharp_i}}$ and $D = C^{\uparrow_{\mathbb{K}^\sharp_i}}$ would contain an attribute $\langle n'', B''\rangle$ such that $B'' \cap F \neq \emptyset$, $B \subseteq B''$, and $n'' > 0$. The attribute $\langle n'', B''\rangle$ would remain in any R-context (either directly or being merged with other attributes) that would further extend the sequence. This follows from the fact that once an attribute has a nonzero flag, it is not removed by any reduction from an R-context (it can be merged together with other attributes during clarification but the nonzero flag remains). Thus, the selection of $\langle n', B'\rangle \in Y^\sharp_i$ would cause that the sequence $\mathbb{K}^\sharp_1 \prec \cdots \prec \mathbb{K}^\sharp_i \prec \mathbb{K}^\sharp_{i+1}$ cannot be extended to a sequence where the last element is an R-context $\mathbb{K}^\sharp_n$ with $X^\sharp_n = E$, meaning that $\langle E, F\rangle$ would not be stored. Altogether, we have shown that for any formal concept the sequence constructed in the proof of Lemma 6.4 is uniquely given. □

7. Complexity and Efficiency Issues

In this section, we inspect worst-case complexity of Algorithm 1 and the underlying operations and present experimental evaluation of its performance compared to other algorithms from the CbO family.

The asymptotic worst-case time complexity of Algorithm 1 is the same as in the case of CbO and FCbO, i.e., $O(|\mathcal{B}(X, Y, I)|\cdot|X|\cdot|Y|^2)$. Indeed, for each formal concept, i.e., for each invocation of COMPUTE, one has to determine the reduced and clarified context which is the argument passed to COMPUTE. This can be done as follows: first, one sorts all attributes in an R-context according to their support. If the support of two different attributes is the same, the attributes can be additionally sorted lexicographically according to sets of objects having those attributes. This can be done in $O(|X|\cdot|Y|\cdot\log|Y|)$ time. Then, attributes that need to be grouped together during clarification can be identified in a single pass through

the set of attributes and the sets of objects having the attributes, i.e. in $O(|X|\cdot|Y|)$ time. Altogether, the R-context is determined in $O(|X|\cdot|Y|\cdot\log|Y|)$ time. Then, Algorithm 1 proceeds as in CbO, i.e., for each attribute, it computes a new closure in $O(|X|\cdot|Y|)$ time and performs the canonicity test in $O(|Y|)$ time. Thus, a single invocation of COMPUTE is done in $O(|X|\cdot|Y|^2)$ time, showing that the asymptotic worst-case time complexity of the algorithm is $O(|\mathcal{B}(X,Y,I)|\cdot|X|\cdot|Y|^2)$. In the case of time delay [10], Algorithm 1 has the same polynomial time delay $O(|Y|^3\cdot|X|)$ as CbO, cf. [17]. The argument remains the same as in the case of CbO.

In order to show the performance of the algorithm compared to other algorithms from the CbO family, we present a set of experiments involving both real-world and artificial datasets and comparison with similar algorithms. All the experiments focus on the total number of computed closures since it is a feature significantly affecting performance of all the algorithms in the CbO family. Table 5 shows counts of closures computed while processing real-world datasets using the CbO, FCbO, and Algorithm 1. Note that the table contains two rows for results of both FCbO and CbO. The rows labeled "ordered" present efficiency of the algorithms if the additional preprocessing step of ordering attributes of input data table according to their support is applied, cf. [13].

From Table 5 it follows that the new algorithm needs to compute considerably less closures than the other algorithms. It seems that this is a general tendency. The tendency is further illustrated by Table 6 and Table 7 containing average counts of computed closures while processing a set of 1,000 artificial data tables. For this experiment we have considered tables of size 50×50, where density of 1s is 10 % and 33 %, respectively, and 1s are distributed approximately normally among attributes.

Table 5. Number of closures computed by selected algorithms from CbO family

	debian tags	anon. web.	mushroom	tic-tac-toe
size	$14,315 \times 475$	$32,710 \times 295$	$8,124 \times 119$	958×29
density	$< 1\,\%$	$1\,\%$	$19\,\%$	$34\,\%$
# concepts	$38,977$	$129,009$	$238,710$	$59,505$
Algorithm 1	$44,221$	$135,925$	$246,181$	$65,567$
FCbO (ordered)	$298,641$	$398,147$	$299,201$	$89,930$
FCbO	$679,911$	$1,475,341$	$426,563$	$128,434$
CbO (ordered)	$960,106$	$785,394$	$1,321,524$	$185,738$
CbO	$12,045,680$	$27,949,552$	$4,006,498$	$221,608$

Table 6. Computed closures in datasets of size 50×50 with 10 % density of 1s

	mean value	standard deviation	median value
CbO	$3,359.88$	505.51	3294
CbO (ordered)	$1,394.08$	78.19	$1,395$
FCbO	860.41	49.17	860
FCbO (ordered)	853.87	47.80	852
Algorithm 1	240.83	8.34	241
# concepts	227.58	6.79	228

Table 7. Computed closures in datasets of size 50×50 with 33 % density of 1s

	mean value	standard deviation	median value
CbO	$332,253.55$	$65,135.75$	$326,097$
CbO (ordered)	$44,074.43$	$6,345.95$	$43,975$
FCbO	$43,787.87$	$6,175.53$	$43,778$
FCbO (ordered)	$32,059.09$	$4,350.26$	$32,057$
Algorithm 1	$25,754.40$	$3,565.85$	$25,776$
# concepts	$24,945.64$	$3,401.93$	$24,958$

Table 8. Ratios of concepts computed multiple times

	debian tags	anon. web.	mushroom	tic-tac-toe
size	$14,315 \times 475$	$32,710 \times 295$	$8,124 \times 119$	958×29
density	$< 1\,\%$	$1\,\%$	$19\,\%$	$34\,\%$
Algorithm 1	0.13	0.05	0.03	0.10
FCbO (ordered)	6.66	2.08	0.25	0.51
FCbO	16.44	10.43	0.78	1.15
CbO (ordered)	23.63	5.08	4.53	2.12
CbO	308.04	215.64	15.78	2.72

Apparently, the new method of computing formal concepts can reduce the total number of computed closures by several orders of magnitude. The factor of improvement depends on many aspects, especially the size of input data. To reduce the influence of this aspect while evaluating algorithms, we use the ratio of concepts computed multiple times (i.e., redundant concepts) to the total number of concepts present in the dataset. Table 8 depicts such ratios for previously discussed real-world datasets. As one can see, the new algorithm while processing *mushroom* dataset computes only 3 % of concepts multiple times. This strongly contrasts with CbO which computes more than fifteen times more concepts than necessary. Furthermore, in case of large and sparse datasets like *anonymous web* and *debian tags* the new algorithm needs to compute only a small fraction of concepts multiple times. This is also a remarkable contrast with the other algorithms since, for instance, CbO computes even hundreds of times more concepts than Algorithm 1.

These tendencies are quite general. For instance, Figure 3 depicts ratios of concepts computed multiple times and their relationship to the number of attributes in the formal context. In this experiment, we have used multiple randomly generated formal contexts having 1,000 objects and various counts of attributes. We have considered data tables with density 5 % and approximately normal distribution of 1s among attributes. Interestingly, it seems that the number of objects has no noticeable impact on the efficiency in terms of concepts computed multiple times as it is shown, e.g., in Figure 4. This figure presents efficiency of algorithms in relationship to the number of objects. In this experiment we have also used artificial datasets and each data table had 100 attributes, various counts of objects, and 1s were distributed approximately normally among attributes with 5 % density. Note that, since CbO (without the preprocessing step) shows a very poor performance, it has been omitted from the chart for the sake of readability.

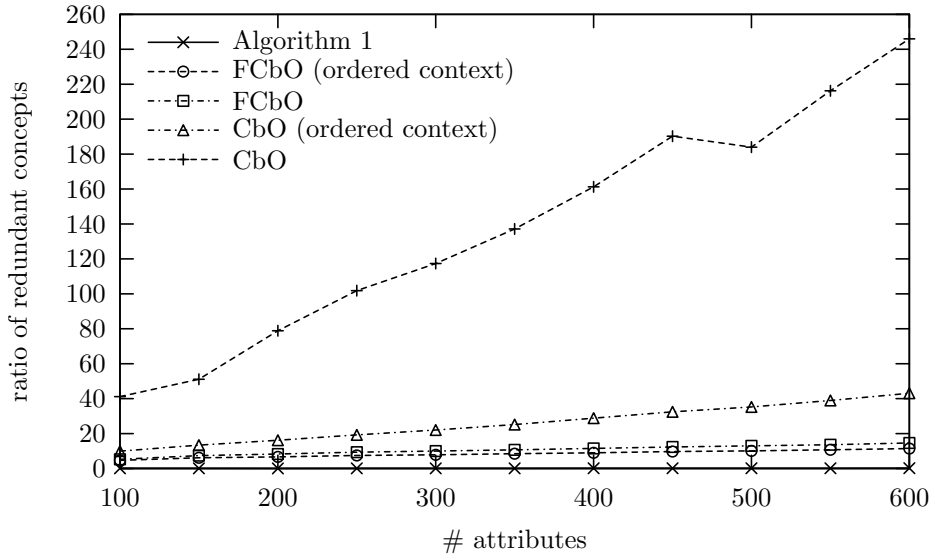

Figure 3. Ratios of concepts computed multiple times and their relationship to the number of attributes

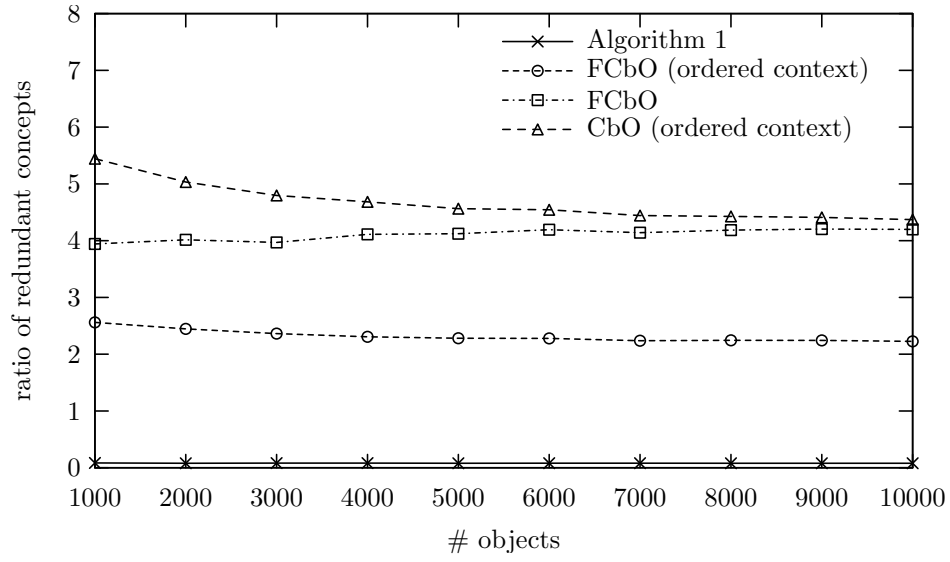

Figure 4. Ratios of concepts computed multiple times and their relationship to the number of objects

References

[1] Agrawal R., Imielinski T., Swami A. N.: Mining association rules between sets of items in large databases, *Proc. ACM Int. Conf. of Management of Data*, 1993, 207–216.

[2] Andrews S.: In-Close, a Fast Algorithm for Computing Formal Concepts, *Supplementary Proceedings of ICCS '09* (S. Rudolph, F. Dau, S. O. Kuznetsov, Eds.), CEUR WS, vol. 483, 2009.

[3] Belohlavek R., Vychodil V.: Discovery of optimal factors in binary data via a novel method of matrix decomposition, *J. Comput. Syst. Sci.*, **76**(1), 2010, 3–20.

[4] Carpineto C., Romano G.: *Concept data analysis. Theory and applications*, J. Wiley, 2004.

[5] Hereth Correia J., Stumme G., Wille R., Wille U.: Conceptual knowledge discovery—a human-centered approach, *Applied Artificial Intelligence*, **17**(3), 2003, 281–302.

[6] Ganter B.: Two basic algorithms in concept analysis, *Proc. ICFCA 2010*, LNCS 5986, 2010, 312–340 (reprint of Technical Report FB4-Preprint No. 831, TH Darmstadt, 1984).

[7] Ganter B., Wille R.: *Formal concept analysis. Mathematical foundations*, Springer, Berlin, 1999.

[8] Goldberg L. A.: *Efficient Algorithms for Listing Combinatorial Structures*, Cambridge University Press, 1993.

[9] Hettich S., Bay S. D.: *The UCI KDD Archive*, University of California, Irvine, School of Information and Computer Sciences, 1999.

[10] Johnson D. S, Yannakakis M., Papadimitriou C. H.: On generating all maximal independent sets, *Information Processing Letters*, **27**(3), 1988, 119–123.

[11] Koester B.: *FooCA – Web Information Retrieval with Formal Concept Analysis*, Verlag Allgemeine Wissenschaft, 2006.

[12] Krajca P., Outrata J., Vychodil V.: Parallel algorithm for computing fixpoints of Galois connections, *Ann. Math. Artif. Intell.*, **59**(2), 2010, 257–272.

[13] Krajca P., Outrata J., Vychodil V.: Advances in algorithms based on CbO, *Proc. CLA 2010* (M. Kryszkiewicz, S. Obiedkov, Eds.), CEUR WS, vol. 672, 2010, 325–337 (`http://ceur-ws.org/Vol-672/paper29.pdf`).

[14] Kuznetsov S. O.: Interpretation on graphs and complexity characteristics of a search for specific patterns, *Automatic Documentation and Mathematical Linguistics*, **24**(1), 1989, 37–45.

[15] Kuznetsov S. O.: A fast algorithm for computing all intersections of objects in a finite semi-lattice (Быстрый алгоритм построения всех пересечений объектов из конечной полурешетки, in Russian), *Automatic Documentation and Mathematical Linguistics*, **27**(5), 1993, 11–21.

[16] Kuznetsov S. O.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. *Proc. PKDD*, 1999, 384–391.

[17] Kuznetsov S. O., Obiedkov S. A.: Comparing performance of algorithms for generating concept lattices, *J. Exp. Theor. Artif. Int.*, **14**, 2002, 189–216.

[18] Kneale W., Kneale M.: *The Development of Logic*, Oxford University Press, USA, 1985.

[19] Lindig C.: Fast concept analysis. *Working with Conceptual Structures–Contributions to ICCS*, 2000, 152–161.

[20] van der Merwe D., Obiedkov S. A., Kourie D. G.: AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. *Proc. ICFCA 2004*, LNAI 2961, 2004, 205–206.

[21] Miettinen P., Mielikäinen T., Gionis A., Das G., Mannila H.: The discrete basis problem. *Proc. PKDD*, 2006, 335–346.

[22] Norris E. M.: An Algorithm for Computing the Maximal Rectangles in a Binary Relation, *Revue Roumaine de Mathématiques Pures et Appliquées*, **23**(2), 1978, 243–250.

[23] Outrata J., Vychodil V.: Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data, *Inf. Sci.*, doi:10.1016/j.ins.2011.09.023.

[24] Pasquier N., Bastide Y., Taouil R., Lakhal L.: Efficient mining of association rules using closed itemset lattices, *Inf. Syst.*, **24**(1), 1999, 25–46.

[25] Snelting G., Tip F.: Reengineering class hierarchies using concept analysis, *ACM Transactions on Programming Languages and Systems*, **22**(3), 2000, 540–582.

[26] Tonella P.: Using a concept lattice of decomposition slices for program understanding and impact analysis, *IEEE Transactions on Software Engineering*, **29**(6), 2003, 495–509.

[27] Wille R.: Restructuring lattice theory: an approach based on hierarchies of concepts. *Ordered Sets,* 1982, 445–470, Dordrecht-Boston.

[28] Zaki M. J., Hsiao C.-J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. *Proc. SIAM DM*, 2002.

[29] Zaki M. J.: Mining non-redundant association rules, *Data Mining and Knowledge Discovery*, **9**, 2004, 223–248.

Inducing decision trees via concept lattices*

Radim Belohlavek[1,3], Bernard De Baets[2], Jan Outrata[3], Vilem Vychodil[3]

[1] Dept. Systems Science and Industrial Engineering,
T. J. Watson School of Engineering and Applied Science, SUNY Binghamton
PO Box 6000, Binghamton, New York 13902–6000, USA
[2] Dept. Appl. Math., Biometrics, and Process Control, Ghent University
Coupure links 653, B-9000 Gent, Belgium
bernard.debaets@ugent.be
[3] Dept. Computer Science, Palacky University, Olomouc
Tomkova 40, CZ-779 00 Olomouc, Czech Republic
{radim.belohlavek, jan.outrata, vilem.vychodil}@upol.cz

Abstract. The paper presents a new method of decision tree induction based on formal concept analysis (FCA). The decision tree is derived using a concept lattice, i.e. a hierarchy of clusters provided by FCA. The idea behind is to look at a concept lattice as a collection of overlapping trees. The main purpose of the paper is to explore the possibility of using FCA in the problem of decision tree induction. We present our method and provide comparisons with selected methods of decision tree induction on testing datasets.

1 Introduction

Decision trees and their induction is one of the most important and thoroughly investigated methods of machine learning [4, 13, 15]. There are many existing algorithms proposed for induction of a decision tree from a collection of records described by attribute vectors. A decision tree forms a model which is then used to classify new records. In general, a decision tree is constructed in a top-down fashion, from the root node to leaves. In each node an attribute is chosen under certain criteria and this attribute is used to split the collection of records covered by the node. The nodes are split until the records have the same value of the decision attribute. The critical point of this general approach is thus the selection of the attribute upon which the records are split. The selection of the splitting attribute is the major concern of the research in the area of decision trees.

The classical methods of attribute selection, implemented in well-known algorithms ID3 and C4.5 [13, 14], are based on minimizing the entropy or information gain, i.e. the amount of information represented by the clusters of records covered by nodes created upon the selection of the attribute. In addition to that, instead of just minimizing the number of misclassified records one can minimize

* Supported by Kontakt 1–2006–33 (Bilateral Scientific Cooperation, project "Algebraic, logical and computational aspects of fuzzy relational modelling paradigms"), by grant No. 1ET101370417 of GA AV ČR, by grant No. 201/05/0079 of the Czech Science Foundation, and by institutional support, research plan MSM 6198959214.

the misclassification and test costs [9]. Completely different solutions are based on involving other methods of machine learning and data mining to the problem of selection of "splitting" attribute. For instance, in [12] the authors use adaptive techniques and computation models to aid a decision tree construction, namely adaptive finite state automata constructing a so-called adaptive decision tree. In our paper, we are going to propose an approach to decision tree induction based on formal concept analysis (FCA), which has been recently utilized in various data mining problems including machine learning via the so-called lattice-based learning techniques. For instance, in [6] authors use FCA in their IGLUE method to select only relevant symbolic (categorical) attributes and transform them to continuous numerical attributes which are better for solving a decision problem by clustering methods (k-nearest neighbor).

FCA produces two kinds of outputs from object-attribute data tables. The first one is called a concept lattice and can be seen as a hierarchically ordered collection of clusters called formal concepts. The second one consists of a non-redundant basis of particular attribute dependencies called attribute implications. A formal concept is a pair of two collections—a collection of objects, called an extent, and a collection of attributes, called an intent. This corresponds to the traditional approach to concepts provided by Port-Royal logic approach.

Formal concepts are denoted by nodes in line diagrams of concept lattices. These nodes represent objects which have common attributes. Nodes in decision trees, too, represent objects which have common attributes. However, one cannot use directly a concept lattice (without the least element) as a decision tree, just because the concept lattice is not a tree in general. See [2] and [1] for results on containment of trees in concept lattices. Moreover, FCA does not distinguish between input and decision attributes. Nevertheless, a concept lattice (without the least element) can be seen as a collection of overlapping trees. Then, a construction of a decision tree can be viewed as a selection of one of these trees. This is the approach we will be interested in in the present paper.

The reminder of the paper is organized as follows. The next section contains preliminaries from decision trees and formal concept analysis. In Section 3 we present our approach of decision tree induction based on FCA. The description of the algorithm is accompanied with an illustrative example. The results of some basic comparative experiments are summarized in Section 4. Finally, Section 5 concludes and outlines several topics of future research.

2 Preliminaries

2.1 Decision trees

A decision tree can be seen as a tree representation of a finitely-valued function over finitely-valued attributes. The function is partially described by assignment of class labels to input vectors of values of input attributes. Such an assignment is usually represented by a table with rows (records) containing values of input attributes and the corresponding class labels. The main goal is to construct a decision tree which represents a function described partially by such a table and,

at the same time, provides the best classification for unseen data (i.e. generalises sufficiently).

Each inner node of a corresponding decision tree is labeled by an attribute, called a decision attribute for this node, and represents a test regarding the values of the attribute. According to the result of the test, records are split into n classes which correspond to n possible outcomes of the test. In the basic setting, the outcomes are represented by the values of the splitting attribute. Leaves of the tree cover the collection of records which all have the same function value (class label). For example, the decision trees in Fig. 1 (right) both represent the function $f : A \times B \times C \to D$ depicted in Fig. 1 (left). This way, a decision tree serves as a model approximating the function partially described by the input data.

A	B	C	$f(A,B,C)$
good	yes	false	yes
good	no	false	no
bad	no	false	no
good	no	true	yes
bad	yes	true	yes

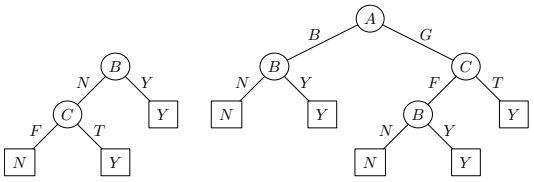

Fig. 1. Two decision trees representing example function f

A decision tree induction problem is the problem of devising a decision tree which approximates well an unknown function described partially by a relatively few records in the table. These records are usually split to two subsets called a training and testing dataset. The training dataset serves as a basis of data from which the decision tree is being induced. The testing dataset is used to evaluate the performance of the decision tree induced by the training dataset.

A vast majority of decision tree induction algorithms uses a strategy of recursive splitting of the collection of records based on selection of decision attributes. This means that the algorithms build the tree from the root to leaves, i.e. in top-down manner. The problem of local optimization is solved in every inner node. Particular algorithms differ by the method solving the optimization problem, i.e. the method of selection of the best attribute to split the records. Traditional criteria of selection of decision attributes are based on entropy, information gain [13] or statistical methods such as χ-square test [10]. The aim is to induce the smallest possible tree (in the number of nodes) which correctly decides training records. The preference of smaller trees follows directly from the Occam's Razor principle according to which the best solution from equally satisfactory ones is the simplest one.

The second problem, which is common to all machine learning methods with a teacher (methods of supervised learning), is the overfitting problem. Overfitting occurs when a model induced from training data behaves well on training data but does not behave well on testing data. A common solution to the overfitting problem used in decision trees is pruning. With pruning, some parts of the

decision tree are omitted. This can either be done during the tree induction process and stop or prevent splitting nodes in some branches before reaching leaves, or after the induction of the complete tree by "post-pruning" some leaves or whole branches. The first way is accomplished by some online heuristics of classification "sufficiency" of the node. For the second way, evaluation of the ability of the tree to classify testing data is used. The simplest criterion for pruning is based on the majority of presence of one function value of records covered by the node.

2.2 Formal concept analysis

In what follows, we summarize basic notions of FCA. An object-attribute data table describing which objects have which attributes can be identified with a triplet $\langle X, Y, I\rangle$ where X is a non-empty set (of objects), Y is a non-empty set (of attributes), and $I \subseteq X \times Y$ is an (object-attribute) relation. Objects and attributes correspond to table rows and columns, respectively, and $\langle x, y\rangle \in I$ indicates that object x has attribute y (table entry corresponding to row x and column y contains $\times$; if $\langle x, y\rangle \notin I$ the table entry contains blank symbol). In the terminology of FCA, a triplet $\langle X, Y, I\rangle$ is called a formal context. For each $A \subseteq X$ and $B \subseteq Y$ denote by $A^{\uparrow}$ a subset of Y and by $B^{\downarrow}$ a subset of X defined by

$$A^{\uparrow} = \{y \in Y \,|\, \text{for each } x \in A:\ \langle x, y\rangle \in I\},$$
$$B^{\downarrow} = \{x \in X \,|\, \text{for each } y \in B:\ \langle x, y\rangle \in I\}.$$

That is, $A^{\uparrow}$ is the set of all attributes from Y shared by all objects from A (and similarly for $B^{\downarrow}$). A formal concept in $\langle X, Y, I\rangle$ is a pair $\langle A, B\rangle$ of $A \subseteq X$ and $B \subseteq Y$ satisfying $A^{\uparrow} = B$ and $B^{\downarrow} = A$. That is, a formal concept consists of a set A (so-called extent) of objects which fall under the concept and a set B (so-called intent) of attributes which fall under the concept such that A is the set of all objects sharing all attributes from B and, conversely, B is the collection of all attributes from Y shared by all objects from A. Alternatively, formal concepts can be defined as maximal rectangles of $\langle X, Y, I\rangle$ which are full of $\times$'s: For $A \subseteq X$ and $B \subseteq Y$, $\langle A, B\rangle$ is a formal concept in $\langle X, Y, I\rangle$ iff $A \times B \subseteq I$ and there is no $A' \supset A$ or $B' \supset B$ such that $A' \times B \subseteq I$ or $A \times B' \subseteq I$.

A set $\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \,|\, A^{\uparrow} = B, B^{\downarrow} = A\}$ of all formal concepts in data $\langle X, Y, I\rangle$ can be equipped with a partial order $\leq$ (modeling the subconcept-superconcept hierarchy, e.g. *dog* $\leq$ *mammal*) defined by

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \text{ iff } A_1 \subseteq A_2 \text{ (iff } B_2 \subseteq B_1). \tag{1}$$

Note that $^{\uparrow}$ and $^{\downarrow}$ form a so-called Galois connection [5] and that $\mathcal{B}(X, Y, I)$ is in fact a set of all fixed points of $^{\uparrow}$ and $^{\downarrow}$. Under $\leq$, $\mathcal{B}(X, Y, I)$ happens to be a complete lattice, called a concept lattice of $\langle X, Y, I\rangle$, the basic structure of which is described by the so-called main theorem of concept lattices [5].

For a detailed information on formal concept analysis we refer to [3, 5] where a reader can find theoretical foundations, methods and algorithms, and applications in various areas.

3 Decision tree induction based on FCA

As mentioned above, a concept lattice without the least element can be seen as a collection of overlapping trees. The induction of a decision tree can be viewed as a selection of one of the overlapping trees. The question is: which tree do we select?

Transformation of input data Before coming to this question in detail, we need to address a particular problem concerning input data. Input data to decision tree induction contains various type of attributes, including yes/no (logical) attributes, categorical (nominal) attributes, ordinal attributes, numerical attributes, etc. On the other hand, Input data to FCA consists of yes/no attributes. Transformation of general attributes to logical attributes is known as conceptual scaling, see [5]. For the sake of simplicity, we consider input data with categorical attributes in our paper and their transformation (scaling) to logical attributes. Decision attributes (class labels) are usually categorical. Note that we need not transform the decision attributes since we do not use them for the concept lattice building step.

Name	body temp.	gives birth	fourlegged	hibernates	mammal
cat	warm	yes	yes	no	yes
bat	warm	yes	no	yes	yes
salamander	cold	no	yes	yes	no
eagle	warm	no	no	no	no
guppy	cold	yes	no	no	no

Name	bt cold	bt warm	gb no	gb yes	fl no	fl yes	hb no	hb yes	mammal
cat	0	1	0	1	0	1	1	0	yes
bat	0	1	0	1	1	0	0	1	yes
salamander	1	0	1	0	0	1	0	1	no
eagle	0	1	1	0	1	0	1	0	no
guppy	1	0	0	1	1	0	1	0	no

Fig. 2. Input data table (top) and corresponding data table for FCA (bottom)

Let us present an example used throughout the presentation of our method. Consider the data table with categorical attributes depicted in Fig. 2 (top). The data table contains sample animals described by attributes *body temperature*, *gives birth*, *fourlegged*, *hibernates* and *mammal*, with the last attribute being the decision attribute (class label). The corresponding data table for FCA with logical attributes obtained from the original ones in an obvious way is depicted in Fig. 2 (bottom).

Step 1 We can now approach the first step of our method of decision tree induction—building the concept lattice. In fact, we do not build the whole lattice. Recall that smaller (lower) concepts result by adding attributes to greater (higher) concepts and, dually, greater concepts result by adding objects to lower

concepts. We can thus imagine the lower neighbor concepts as refining their parent concept. In a decision tree the nodes cover some collection of records and are split until the covered records have the same value of class label. The same applies to concepts in our approach: we need not split concepts which cover objects having the same value of class label. Thus, we need an algorithm which generates a concept lattice from the greatest concept (which covers all objects) and iteratively generates lower neighbor concepts.

For this purpose, we can conveniently use the essential ideas of Lindig's NextNeighbor algorithm [8]. NextNeighbor efficiently generates formal concepts together with their subconcept-superconcept hierarchy. Our method, which is a modification of NextNeighbor, differs from the ordinary NextNeighbor in two aspects. First, as mentioned above, we do not compute lower neighbor concepts of a concept which covers objects with the same class label. Second, unlike NextNeighbor, we do not build the ordinary concept hierarchy by means of a covering relation. Instead, we are skipping some concepts in the hierarchy. That is, a lower neighbor concept c of a given concept d generated by our method, can in fact be a concept for which there exists an intermediate concept between c and d. This is accomplished by a simple modification of NextNeighbor algorithm.

NextNeighbor The NextNeighbor algorithm builds the concept lattice by iteratively generating the neighbor concepts of a concept $\langle A, B\rangle$, either top-down the lattice by adding new attributes to concept intents or bottom-up by adding new objects to concept extents. We follow the top-down approach. The algorithm is based on the fact that a concept $\langle C, D\rangle$ is a neighbor of a given concept $\langle A, B\rangle$ if D is generated by $B \cup \{y\}$, i.e. $D = (B \cup \{y\})^{\downarrow\uparrow}$, where $y \in Y - B$ is an attribute such that for all attributes $z \in D - B$ it holds that $B \cup \{z\}$ generates the same concept $\langle C, D\rangle$ [8], i.e.

$$\begin{aligned}&(Next)Neighbors\ of\ \langle A, B\rangle = \\ &\quad \{\langle C, D\rangle \mid D = (B \cup \{y\})^{\downarrow\uparrow}, y \in Y - B\ such\ that \\ &\qquad (B \cup \{z\})^{\downarrow\uparrow} = D\ for\ all\ z \in D - B\}.\end{aligned}$$

Our modification From the monotony of the (closure) operator forming a formal concept it follows that a concept $\langle C, D\rangle$ is not a neighbor of the concept $\langle A, B\rangle$ if there exists an attribute $z \in D - B$ such that $B \cup \{z\}$ generates a concept between $\langle A, B\rangle$ and $\langle C, D\rangle$. This is what our modification consists in. Namely, we mark as (different) neighbors all concepts generated by $B \cup \{y\}$ for $y \in Y - B$, even those for which there exists a concept in between, i.e.

$$\begin{aligned}&(Our)Neighbors\ of\ \langle A, B\rangle = \\ &\quad \{\langle C, D\rangle \mid D = (B \cup \{y\})^{\downarrow\uparrow}, y \in Y - B\}.\end{aligned}$$

It is easy to see that our modification does not alter the concept lattice and the overall hierarchy of concepts, cf. NextNeighbor [8].

The reason for this modification is that we have to record as neighbors of a concept $\langle A, B\rangle$ all the concepts which are generated by the collection of attributes

B with one additional attribute. In the resulting decision tree the addition of a (logical) attribute to a concept means making a decision on the corresponding categorical attribute in the tree node corresponding to the concept. Due to lack of space we postpone a pseudocode of the algorithm of Step 1 to the full version of the paper. Part of the concept lattice built from data table in Fig. 2 (bottom), with our new neighbor relationships drawn by dashed lines, is depicted in Fig. 3.

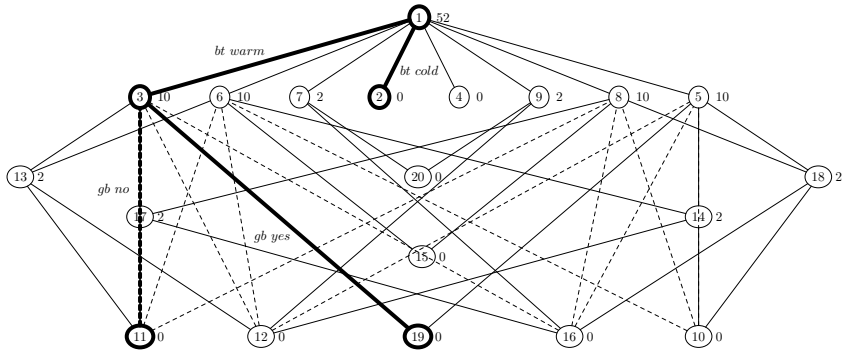

Fig. 3. Part of the concept lattice and tree of concepts (solid) of data table in Fig. 2

Step 2 The second step of our method is the selection of a tree of concepts from the part of the concept lattice built in the first step. First, we calculate for each concept $c = \langle A, B \rangle$ the number L_c of all of its lower concepts. Note that each lower concept is counted for each different attribute added to the concept c, cf. our modification of concept neighbor relation. For instance, if a concept $d = \langle C, D \rangle$ is generated from concept c by adding either attribute x or attribute y (i.e. $D = (B \cup \{x\})^{\downarrow\uparrow}$ or $D = (B \cup \{y\})^{\downarrow\uparrow}$, respectively), the concept d is counted twice and L_c is increased by two.

Next, we select a tree of concepts from the part of the concept lattice by iteratively going from the greatest concept (generated by no attributes or, equivalently, by all objects) to minimal concepts. The selection is based on the number L_c of lower concepts of the currently considered concept c (recall that L_c is not the number of lower concepts of c in common sense, cf. the computation of L_c above which is due to our modification of concept neighbor relation).

The root node of the tree is always the greatest concept. Then, for each tree node/concept c we define collections $\mathcal{N}_c^a$ of concepts, which will be candidate collections of children nodes/concepts of c in the resulting selected tree. $\mathcal{N}_c^a$ is a collection of lower neighbor concepts of c such that (a) each concept d in $\mathcal{N}_c^a$ is generated from concept c by adding a (logical) attribute transformed from the categorical attribute a (recall that logical attributes stand for values of categorical attributes) and (b) $\mathcal{N}_c^a$ contains the concept d for every logical attribute transformed from categorical attribute a. There (1) may exist, and

usually exists, more than one such collection $\mathcal{N}_c^a$ of neighbor concepts of concept c, for more that one categorical attribute a, but, on the other side (2) there may exist no such collection.

(1) In this case we choose from the several collections $\mathcal{N}_c^a$ of neighbor concepts of concept c the collection containing a concept d with the minimal number L_d of its lower concepts. Furthermore, if there is more than one such neighbor concept, in different collections, we choose the collection containing the concept which covers the maximal number of objects/records. It is important to note that this point is the only non-deterministic point in our method since there still can be more than one neighbor concepts having equal minimal number of lower concepts and covering equal maximal number of objects. In that case we choose one of the collections $\mathcal{N}_c^a$ of neighbor concepts arbitrarily.

(2) This case means that in every potential collection $\mathcal{N}_c^a$ there is missing at least one neighbor concept generated by some added (logical) attribute transformed from categorical attribute a, i.e. $\mathcal{N}_c^a$ does not satisfy the condition (b). We solve this situation by substituting the missing concepts by (a copy of) the least concept $\langle Y^{\downarrow}, Y\rangle$ generated by all attributes (or, equivalently, no objects). The least concept is a common and always existing subconcept of all concepts in a concept lattice and usually covers no objects/records (but need not!).

Finally, an edge between concept c and each neighbor concept from the chosen collection $\mathcal{N}_c^a$ is created in the resulting selected tree. The edge is labeled by the added logical attribute. Again, we postpone a pseudocode of the algorithm of Step 2 to the full version of the paper.

To illustrate the previuos description, let us consider the example of a part of the concept lattice in Fig. 3. The concepts are denoted by a circled number and the number of lower concepts is written to the right of every concept. We select the tree of concepts as follows. The root node of the tree is the greatest concept 1. As children nodes of the root node are selected concepts 2 and 3 since they form a collection $\mathcal{N}_1^{body\ temp.}$ of all lower neighbor concepts generated by both added (logical) attributes *bt cold* and *bt warm*, respectively, transformed from the categorical attribute *body temp.*. Note that we could have chosen the collection $\mathcal{N}_1^{gives\ birth}$ instead of collection $\mathcal{N}_1^{body\ temp.}$, but since both concepts 2 from $\mathcal{N}_1^{body\ temp.}$ and 4 from $\mathcal{N}_1^{gives\ birth}$ have the equal minimal number L_2 and L_4 of lower concepts and both cover the equal maximal number of objects/records, we have chosen the collection $\mathcal{N}_1^{body\ temp.}$ arbitrarily, according to case (1) from the description. The edges of the selected tree are labeled by the corresponding logical attributes. Similarly, the children nodes of concept 3 will be concepts 11 and 19, and this is the end of the tree selection step since concepts 4, 11 and 19 have no lower neighbors. The resulting tree of concepts is depicted in Fig. 3 with solid lines.

Step 3 The last step (third one) of our method is the transformation of the tree of concepts into a decision tree. A decision tree has in its every node the chosen categorical attribute on which the decision is made and the edges from the node are labeled by the possible values of the attribute. The leaves are labeled by

class label(s) of covered records. In the tree of concepts the logical attributes transformed from (and standing for the values of) the categorical attribute are in the labels of edges connecting concepts. Hence the transformation of the tree of concepts into a decision tree is simple: edges are relabeled to the values of the categorical attribute, inner concepts are labeled by the corresponding categorical attributes, and leaves are labelled by class label(s) of covered objects/records.

The last problem to solve is multiple different class label(s) of covered records in tree leaves. This can happen for several reasons, for example the presence of conflicting records in input data differing in class label(s) only (which can result for instance from class labelling mistakes or from selecting a subcollection of attributes from original larger data) or pruning the complete decision tree as a strategy to the overfitting problem. Common practice for dealing with multiple different target class label(s) is as simple as picking the major class label value(s) as the target classification of records covered by leave node and we adopt this solution. A special case are leave nodes represented by (a copy of) the least concept (which comes from the possibility (2) in Step 2), since the least concept usually covers no objects/records. These nodes are labelled by the class label(s) of their parent nodes.

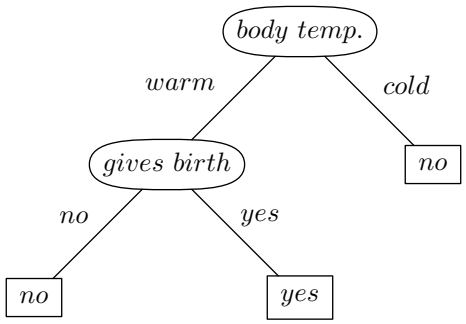

Fig. 4. The decision tree of input data in Fig. 2

The resulting decision tree of input data in Fig. 2 (top) transformed from the tree of concepts in Fig. 3 is depicted in Fig. 4.

Let us now briefly discuss the problem of overfitting. A traditional solution to overfitting problem, i.e. pruning, suggests not to include all nodes down to leaves as a part of the decision tree. One of the simplest criteria for this is picking a threshold percentage ratio of major class label value(s) in records covered by a node. Alternatively, one can use the entropy measure to decide whether the node is sufficient and need not be split. Note that several other possibilities exist. In all cases, the constraint can be applied as early as selecting the concepts to the tree, i.e. pruning can be done during decision tree induction process in our method. No pruning method is considered in this paper.

4 Comparison with other algorithms

The asymptotic time complexity of the presented algorithm is given by the (part of the) concept lattice building step since this step is the most time demanding. The concepts are computed by a modified Lindig's NextNeighbor algorithm. Since the modification does not alter the asymptotic time complexity, the overall asymptotic complexity of our method is equal to that of Lindig's NextNeighbor algorithm, namely $O(|X||Y|^2|L|)$. Here, $|X|$ is the number of input records, $|Y|$ is the number of (logical) attributes and $|L|$ is the size of the concept lattice, i.e. the number of all formal concepts.

However, for the decision tree induction problem, accuracy, i.e. the percentage of correctly and incorrectly decided records from both training and testing dataset, is more important than time complexity. We performed preliminary experiments and compared our method to reference algorithms ID3 and C4.5. We implemented our method in C language. ID3 and C4.5 were borrowed and run from the Weka[4] (Waikato Environment for Knowledge Analysis [16]), a software package which aids the development of and contains implementations of several machine learning and data mining algorithms in Java. Default Weka's parameters were used for the two algorithms and pruning was turned off where available.

Table 1. Characteristics of datasets used in experiments

Dataset	No. of attributes	No. of records	Class distribution
breast-cancer	6	138	100/38
kr-vs-kp	14	319	168/151
mushroom	10	282	187/95
vote	8	116	54/62
zoo	9	101	41/20/5/13/4/8/10

The experiments were done on selected public real-world datasets from UCI machine learning repository [7]. The selected datasets are from different areas (medicine, biology, zoology, politics and games) and all contain only categorical attributes with one class label. The datasets were cleared of records containing missing values and actually, we selected subcollections of less-valued attributes of each dataset and subcollections of records of some datasets, due to computational time of repeated executions on the same dataset. The basic characteristics of the datasets are depicted in Tab. 1. The results of averaging 10 executions of the 10-Fold Stratified Cross-validation test (which gives total of 100 executions for each algorithm over each dataset) are depicted in Tab. 2. The table shows average percentage rates of correct decisions for both training (upper item in the table cell) and testing (lower item) dataset part, for each compared algorithm and dataset. We can see that our FCA based decision tree induction method outperforms C4.5 on all datasets, with the exception of mushroom, by 2 – 4 %,

[4] Weka is a free software available at `http://www.cs.waikato.ac.nz/ml/weka/`

on both training and testing data, and gains almost identical results as ID3, again on all datasets except mushroom, on training data, but slightly outperforming ID3 on testing data by about 1 %. On mushroom dataset, which is quite sparse comparing to the other datasets, our method is little behind ID3 and C4.5 on training data, but, however, almost equal on testing data. The datasets vote and zoo are more dense than the other datasets and also contain almost no conflicting records, so it seems that the FCA based method could give better results that traditional, entropy based, methods on clear dense data. However, more experiments on additional datasets are needed to approve this conclusion.

Table 2. Percentage correct rates for datasets in Tab. 1

training % *testing %*	breast-cancer	kr-vs-kp	mushroom	vote	zoo
FCA based	88.631 79.560	84.395 74.656	96.268 96.284	97.528 90.507	98.019 96.036
ID3	88.630 75.945	84.674 74.503	97.517 96.602	97.528 89.280	98.019 95.036
C4.5	86.328 79.181	82.124 72.780	97.163 96.671	94.883 86.500	96.039 92.690

Due to lack of space only the basic experiments are presented. More comparative tests on additional datasets with additional various machine learning algorithms like Naive Bayes classification or Artificial Neural Networks trained by back propagation [11], including training and testing time measuring, are postponed to the full version of the paper. However, the first preliminary experiments show that our simple FCA based method is promising in using FCA in the decision tree induction problem.

The bottleneck of the method could be performance, the total time of tree induction, but once one already has the (whole) concept lattice of input data, then the tree selection is very fast. This draws a possible usage and perspective of the method: decision making from already available concept lattices. The advantage of our method over other methods is the conceptual information hidden in tree nodes (note that they are in fact formal concepts). The attributes in concept intents are the attributes common to all objects/records covered by the concept/tree node, which might be usefull information for furher exploration, application and interpretation of the decision tree. This type of information is not (directly) available by other methods, for instance classical entropy based.

5 Conclusion and topics of future research

We have presented a simple novel method of decision tree induction by selection of the tree of concepts from a concept lattice. The criterion of choosing an attribute based on which the node of the tree is split is determined by the number of all lower concepts of the concept corresponding to the node. The approach interconnects areas of decision trees and formal concept analysis. We have also

presented some comparison to classical decision tree algorithms, namely ID3 and C4.5, and have seen that our method compares quite well and surely deserves more attention. Topics for future research include:

- Explore the possibility to compute a smaller number of formal concepts from which the nodes of a decision tree is constructed. Or, the possibility to compute right the selected concepts only.
- The problems of overfitting in data and uncomplete data, i.e. data having missing values for some attributes in some records.
- Incremental updating of induced decision trees via incremental methods of building concept lattices.

References

1. Belohlavek R., De Baets B., Outrata J., Vychodil V.: Trees in Concept Lattices. In: Torra V., Narukawa Y.,Yoshida Y. (Eds.): Modeling Decisions for Artificial Intelligence: 4th International Conference, MDAI 2007, *Lecture Notes in Artificial Intelligence* **4617**, pp. 174–184, Springer-Verlag, Berlin/Heidelberg, 2007.
2. Belohlavek R., Sklenar V.: Formal concept analysis constrained by attribute-dependency formulas. In: B. Ganter and R. Godin (Eds.): ICFCA 2005, *Lecture Notes in Computer Science* **3403**, pp. 176–191, Springer-Verlag, Berlin/Heidelberg, 2005.
3. Carpineto C., Romano G.: *Concept Data Analysis. Theory and Applications.* J. Wiley, 2004.
4. Dunham M. H.: *Data Mining. Introductory and Advanced Topics.* Prentice Hall, Upper Saddle River, NJ, 2003.
5. Ganter B., Wille R.: *Formal Concept Analysis. Mathematical Foundations.* Springer, Berlin, 1999.
6. Mephu Nguifo E., Njiwoua P.: IGLUE: A lattice-based constructive induction system. *Intell. Data Anal.* **5**(1), pp. 73–91, 2001.
7. Newman D. J., Hettich S., Blake C. L., Merz C. J.: *UCI Repository of machine learning databases*, [`http://www.ics.uci.edu/~mlearn/MLRepository.html`], Irvine, CA: University of California, Department of Information and Computer Science, 1998.
8. Lindig C.: Fast concept analysis. In: Stumme G.: *Working with Conceptual Structures – Contributions to ICCS 2000.* Shaker Verlag, Aachen, 2000, 152–161.
9. Ling Ch. X., Yang Q., Wang J., Zhang S.: Decision Trees with Minimal Costs. *Proc. ICML 2004.*
10. Mingers J.: Expert systems – rule induction with statistical data. *J. of the Operational Research Society.*, **38**, pp. 39–47, 1987.
11. Mitchell T.M.: *Machine Learning.* McGraw-Hill, 1997.
12. Pistori H., Neto J. J.: Decision Tree Induction using Adaptive FSA. *CLEI Electron. J.* **6**(1), 2003.
13. Quinlan J. R.: *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
14. Quinlan J. R.: Learning decision tree classifiers. *ACM Computing Surveys*, **28**(1), 1996.
15. Tan P.-N., Steinbach M., Kumar V.: *Introduction to Data Mining.* Addison Wesley, Boston, MA, 2006.
16. Witten I. H., Frank E.: *Data Mining: Practical machine learning tools and techniques, 2nd Edition.* Morgan Kaufmann, San Francisco, 2005.

Inducing decision trees via concept lattices

Jan Outrata*
Dept. Computer Science, Palacky University, Olomouc
Tomkova 40, CZ-779 00 Olomouc, Czech Republic
jan.outrata@upol.cz

Abstract

The paper presents a new machine learning method of decision tree induction based on formal concept analysis (FCA). FCA is a data mining technique the output of which is a hierarchical structure of clusters extracted from data describing objects by attributes. The decision tree is derived using the structure of clusters (called concept lattice). The idea behind is to look at a concept lattice as a collection of overlapping trees. The main purpose of the paper is to explore the possibility of using FCA in the problem of decision tree induction. We present our method and provide comparisons with selected methods of decision tree induction and machine learning on testing datasets.

1 Introduction

Decision trees and their induction is one of the most important and thoroughly investigated methods of machine learning [Dunham, 2003; Quinlan, 1993; Tan, Steinbach and Kumar, 2006]. Machine learning is one of the major fields in artificial intelligence which concerns with the development of methods and techniques that allow machines to "learn". Allowing machines to perform difficult taks of human reasoning, the methods of machine learning can be applied in several areas of system sciences including intelligent control systems, adaptive systems, robotics and even cybernetics. Decision trees, being an efficient classification models of data, support machine learning in the problem of decision making.

A decision tree forms a model which is then used to classify new records. There are many existing algorithms proposed for induction of a decision tree from a collection of records described by attribute vectors. In general, a decision tree is constructed in a top-down fashion, from the root node to leaves. In each node an attribute is chosen under certain criteria and this attribute is used to split the collection of records covered by the node. The nodes are split until the records have the same value of the decision attributes (often called class labels). The critical point of this general approach is thus the selection of the attribute upon which the records are split. The selection of the splitting attribute is the major concern of the research in the area of decision trees. Decision trees have also more descriptive names of classification trees or regression trees in the case of discrete or continuous class labels, respectively.

The classical methods of attribute selection, implemented in well-known algorithms ID3 and C4.5 [Quinlan, 1993; 1996], are based on minimizing the entropy or information gain, i.e. the amount of information represented by the clusters of records covered by nodes created upon the selection of the attribute. However, these methods use statistics only, without any particular view on data, and thus are limited in efficiency of the created model. Completely different solutions are based on involving other methods of machine learning and data mining. For instance, in [Pistori and Neto, 2003] the authors use adaptive techniques and computation models to aid a decision tree construction, namely adaptive finite state automata constructing a so-called adaptive decision tree.

In our paper, we are going to propose an approach to decision tree induction based on formal concept analysis (FCA), which has been recently utilized in various data mining problems including machine learning, via so-called lattice-based learning techniques [Fu, Fu, Njiwoua and Mephu Nguifo, 2004; Kuznetsov, 2004]. For instance, in [Mephu Nguifo and Njiwoua, 2001] authors use FCA in their IGLUE method to select only relevant symbolic (categorical) attributes and transform them to continuous numerical attributes which are better for solving a decision problem by clustering methods (k-nearest neighbor). However, we are going to use FCA more directly, employing a conceptual view on data which FCA offers, since we believe that this can help to create a more fitted model of data.

FCA produces two kinds of outputs from data tables consisting of records (objects in terminology of FCA) described by attributes. The first one is called

*Supported by Kontakt 1–2006–33 (Bilateral Scientific Cooperation, project "Algebraic, logical and computational aspects of fuzzy relational modelling paradigms"), by grant No. 1ET101370417 of GA AV ČR, by grant No. 201/05/0079 of the Czech Science Foundation, and by institutional support, research plan MSM 6198959214.

a concept lattice and can be seen as a hierarchically ordered collection of clusters called formal concepts. The second one consists of a non-redundant basis of particular attribute dependencies called attribute implications. A formal concept is a formalization of a notion of concept in human reasoning, defined as a pair of two collections—a collection of objects, called an extent, and a collection of attributes, called an intent. This corresponds to the traditional approach to concepts provided by Port-Royal logic approach.

Formal concepts represent objects/records which have common attributes. Nodes in decision trees, too, represent records which have common attributes. However, one cannot use directly a concept lattice (without the least element) as a decision tree, just because the concept lattice is not a tree in general. See [Belohlavek and Sklenar, 2005] and [Belohlavek, De Baets, Outrata and Vychodil, 2007] for results on containment of trees in concept lattices. Moreover, FCA does not distinguish between input and decision attributes. Nevertheless, a concept lattice (without the least element) can be seen as a collection of overlapping trees. Then, a construction of a decision tree can be viewed as a selection of one of these trees. This is the approach we will be interested in in the present paper.

The reminder of the paper is organized as follows. The next section contains preliminaries from decision trees and formal concept analysis. In Section 3 we present our approach of decision tree induction based on FCA. The description of the algorithm is accompanied with an illustrative example. The results of some basic comparative experiments are summarized in Section 4. Finally, Section 5 concludes and outlines several topics of future research.

2 Preliminaries

2.1 Decision trees

A decision tree can be seen as a tree representation of a finitely-valued function over finitely-valued attributes. The function is partially described by assignment of class label(s) to input vectors of values of input attributes. Such an assignment is usually represented by a table with rows (records) containing values of input attributes and the corresponding class label(s). The main goal is to construct a decision tree which represents a function described partially by such a table and, at the same time, provides the best classification for unseen data (i.e. generalises sufficiently).

Each inner node of a corresponding decision tree is labeled by an attribute, called a decision attribute for this node, and represents a test regarding the values of the attribute. According to the result of the test, records are split into n classes which correspond to n possible outcomes of the test. In the basic setting, the outcomes are represented by the values of the splitting attribute. Leaves of the tree cover the collection of records which all have the same function value (class label). For example, the decision trees in Fig. 1 (bottom) both represent the function $f : A \times B \times C \to D$ depicted in Fig. 1 (top). This way, a decision tree

A	B	C	$f(A,B,C)$
good	yes	false	yes
good	no	false	no
bad	no	false	no
good	no	true	yes
bad	yes	true	yes

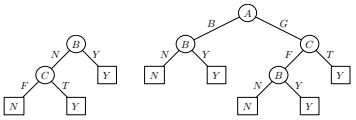

Figure 1: Two decision trees (bottom) representing example function f (top)

serves as a model approximating the function partially described by the input data.

A decision tree induction problem is the problem of devising a decision tree which approximates well an unknown function described partially by a relatively few records in the table. These records are usually split to two subsets called a training and testing dataset. The training dataset serves as a basis of data from which the decision tree is being induced. The testing dataset is used to evaluate the performance of the decision tree induced by the training dataset.

A vast majority of decision tree induction algorithms uses a strategy of recursive splitting of the collection of records based on selection of decision attributes. This means that the algorithms build the tree from the root to leaves, i.e. in top-down manner. The problem of local optimization is solved in every inner node. Particular algorithms differ by the method solving the optimization problem, i.e. the method of selection of the best attribute to split the records. Traditional criteria of selection of decision attributes are based on entropy, information gain [Quinlan, 1993] or statistical methods such as χ-square test [Mingers, 1987]. The aim is to induce the smallest possible tree (in the number of nodes) which correctly decides training records. The preference of smaller trees follows directly from the Occam's Razor principle according to which the best solution from equally satisfactory ones is the simplest one.

2.2 Formal concept analysis

In what follows, we summarize basic notions of FCA. An object-attribute data table describing which objects have which attributes can be identified with a triplet $\langle X, Y, I\rangle$ where X is a non-empty set of objects, Y is a non-empty set of attributes, and $I \subseteq X \times Y$ is an object-attribute relation. Objects and attributes correspond to table rows and columns, respectively, and $\langle x, y\rangle \in I$ indicates that object x has attribute y (table entry corresponding to row x and column y contains $\times$; if $\langle x, y\rangle \notin I$ the table entry contains blank symbol). In the terminology of FCA, a triplet $\langle X, Y, I\rangle$ is called a formal context. For each $A \subseteq X$ and $B \subseteq Y$ denote by $A^{\uparrow}$ a subset of Y and by $B^{\downarrow}$ a subset of X defined by

$$\begin{aligned} A^{\uparrow} &= \{y \in Y \mid \text{for each } x \in A:\ \langle x, y\rangle \in I\}, \\ B^{\downarrow} &= \{x \in X \mid \text{for each } y \in B:\ \langle x, y\rangle \in I\}. \end{aligned}$$

That is, $A^{\uparrow}$ is the set of all attributes from Y shared by all objects from A (and similarly for $B^{\downarrow}$). A formal concept in $\langle X, Y, I\rangle$ is a pair $\langle A, B\rangle$ of $A \subseteq X$ and $B \subseteq Y$ satisfying $A^{\uparrow} = B$ and $B^{\downarrow} = A$. That is, a formal concept consists of a set A (so-called extent) of objects which fall under the concept and a set B (so-called intent) of attributes which fall under the concept such that A is the set of all objects sharing all attributes from B and, conversely, B is the collection of all attributes from Y shared by all objects from A. Alternatively, formal concepts can be defined as maximal rectangles of $\langle X, Y, I\rangle$ which are full of $\times$'s: For $A \subseteq X$ and $B \subseteq Y$, $\langle A, B\rangle$ is a formal concept in $\langle X, Y, I\rangle$ iff $A \times B \subseteq I$ and there is no $A' \supset A$ or $B' \supset B$ such that $A' \times B \subseteq I$ or $A \times B' \subseteq I$. Formal concepts represent clusters hidden in object-attribute data.

A set $\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \,|\, A^{\uparrow} = B, B^{\downarrow} = A\}$ of all formal concepts in data $\langle X, Y, I\rangle$ can be equipped with a partial order $\leq$ modeling the subconcept-superconcept hierarchy, e.g. *dog* $\leq$ *mammal*, defined by $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ iff $A_1 \subseteq A_2$ (iff $B_2 \subseteq B_1$). Note that $^{\uparrow}$ and $^{\downarrow}$ form a so-called Galois connection [Ganter and Wille, 1999] and that $\mathcal{B}(X, Y, I)$ is in fact a set of all fixed points of $^{\uparrow}$ and $^{\downarrow}$. Under $\leq$, $\mathcal{B}(X, Y, I)$ happens to be a complete lattice, called a concept lattice of $\langle X, Y, I\rangle$, the basic structure of which is described by the so-called main theorem of concept lattices [Ganter and Wille, 1999].

Theorem 1 (1) *The set $\mathcal{B}(X, Y, I)$ is under $\leq$ a complete lattice where the infima and suprema are given by*

$$\bigwedge_{j\in J}\langle A_j, B_j\rangle = \langle \bigcap_{j\in J} A_j, (\bigcup_{j\in J} B_j)^{\downarrow\uparrow}\rangle,$$
$$\bigvee_{j\in J}\langle A_j, B_j\rangle = \langle (\bigcup_{j\in J} A_j)^{\uparrow\downarrow}, \bigcap_{j\in J} B_j\rangle.$$

(2) *Moreover, an arbitrary complete lattice $\mathbf{V} = \langle V, \leq\rangle$ is isomorphic to $\mathcal{B}(X, Y, I)$ iff there are mappings $\gamma : X \to V$, $\mu : Y \to V$ such that*

(i) *$\gamma(X)$ is $\bigvee$-dense in V, $\mu(Y)$ is $\bigwedge$-dense in V;*

(ii) *$\gamma(x) \leq \mu(y)$ iff $\langle x, y\rangle \in I$.*

For a detailed information on formal concept analysis we refer to [Carpineto and Romano, 2004; Ganter and Wille, 1999] where a reader can find theoretical foundations, methods and algorithms, and applications in various areas.

3 Decision tree induction based on FCA

As mentioned above, a concept lattice without the least element can be seen as a collection of overlapping trees. The induction of a decision tree can be viewed as a selection of one of the overlapping trees. The question is: which tree do we select?

Transformation of input data Before coming to this question in detail, we need to address a particular problem concerning input data. Input data to decision tree induction contains various type of attributes, including yes/no (logical) attributes, categorical (nominal) attributes, ordinal attributes, numerical attributes, etc. On the other hand, Input data to FCA consists of yes/no attributes. Transformation of general attributes to logical attributes is known as conceptual scaling, see [Ganter and Wille, 1999]. For the sake of simplicity, we consider input data with categorical attributes in our paper and their transformation (scaling) to logical attributes. Decision attributes (class labels) are usually categorical. Note that we need not transform the decision attributes since we do not use them for the concept lattice building step.

Let us present an example used throughout the presentation of our method. Consider the data table with categorical attributes depicted in Fig. 2 (top). The data table contains sample animals described by attributes *body temperature*, *gives birth*, *fourlegged*, *hibernates* and *mammal*, with the last attribute being the decision attribute (class label). The corresponding data table for FCA with logical attributes obtained from the original ones in an obvious way is depicted in Fig. 2 (bottom).

Step 1 We can now approach the first step of our method of decision tree induction—building the concept lattice. In fact, we do not build the whole lattice. Recall that smaller (lower) concepts result by adding attributes to greater (higher) concepts. We can thus imagine the lower neighbor concepts as refining their parent concept. In a decision tree the nodes cover some collection of records and are split until the covered records have the same value of class label(s). The same applies to concepts in our approach: we need not split concepts which cover objects having the same value of class label(s). Thus, we need an algorithm which generates a concept lattice from the greatest concept (which covers all objects) and iteratively generates lower neighbor concepts. Such an algorithm is Lindig's NextNeighbor [Lindig, 2000], for instance.

In our method we further modify the concept lattice building algorithm in two aspects. First, as mentioned above, we do not compute lower neighbor concepts of a concept which covers objects with the same class label(s). Second, we do not build the ordinary concept hierarchy by means of a covering relation, but instead, we are skipping some concepts in the hierarchy. That is, a lower neighbor concept c of a given concept d generated by our method, can in fact be a concept for which there exists an intermediate concept between c and d.

The reason for this modification is that we have to record as neighbors of a concept $\langle A, B\rangle$ all the concepts which are generated by the collection of attributes B with one additional attribute. In the resulting decision tree the addition of a (logical) attribute to a concept means making a decision on the corresponding categorical attribute in the tree node corresponding to the concept. Due to lack of space we postpone a pseudocode of the algorithm of Step 1 to the full version of the paper. Part of the concept lattice built from data table in Fig. 2 (bottom), with our new neighbor relationships drawn by dashed lines, is depicted in Fig. 3.

Name	body temp.	gives birth	fourlegged	hibernates	mammal
cat	warm	yes	yes	no	yes
bat	warm	yes	no	yes	yes
salamander	cold	no	yes	yes	no
eagle	warm	no	no	no	no
guppy	cold	yes	no	no	no

Name	bt cold	bt warm	gb no	gb yes	fl no	fl yes	hb no	hb yes	mammal
cat	0	1	0	1	0	1	1	0	yes
bat	0	1	0	1	1	0	0	1	yes
salamander	1	0	1	0	0	1	0	1	no
eagle	0	1	1	0	1	0	1	0	no
guppy	1	0	0	1	1	0	1	0	no

Figure 2: Input data table (top) and corresponding data table for FCA (bottom)

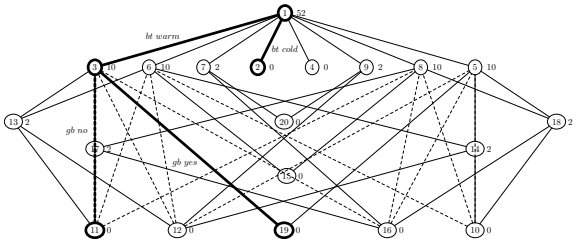

Figure 3: Part of the concept lattice and tree of concepts (solid) of data table in Fig. 2

Step 2 The second step of our method is the selection of a tree of concepts from the part of the concept lattice built in the first step. First, we calculate for each concept c the number L_c of all of its lower concepts. Note that each lower concept is counted for each different attribute added to the concept c, cf. our modification of concept neighbor relation.

Furthermore, for a concept c and a categorical attribute a we define a collection $\mathcal{N}_c^a$ containing for each logical attribute z transformed from a the lower neighbor concept of c generated from c by adding z, if such a neighbor concept exists, or otherwise, (a copy of) the least concept $\langle Y^{\downarrow}, Y\rangle$ generated by all attributes, with $L_{\langle Y^{\downarrow},Y\rangle} = \infty$.

Now, we select a tree of concepts from the part of the concept lattice by iteratively going from the greatest concept (generated by all objects) to minimal concepts. The selection is based on the number L_c of lower concepts of the currently considered concept c.

(1) The root node of the tree is always the greatest concept. This is the starting point of the tree selection.

(2) For each tree node/concept c we select the children nodes/concepts in the selected tree as follows. First, from all lower neighbor concepts of c we select the concept d with the minimal number L_d of its lower concepts. Furthermore, if there is more than one such neighbor concept, generated from c by adding logical attributes transformed from different categorical attributes, we select the one covering the maximal number of objects/records. However, if there is still more than one such concept, we select one of them arbitrarily. Selecting the neighbor concept, which is generated from c by adding a (logical) attribute transformed from the categorical attribute a, means selecting a for a decision attribute. Then, the children nodes/concepts of c are going to be the concepts from the collection $\mathcal{N}_c^a$.

(3) Finally, an edge between concept c and each concept from the collection $\mathcal{N}_c^a$ is created in the resulting selected tree. The edge is labeled by the logical attribute added in the concept. This means drawing result possibilities of a decision test on attribute a.

Again, we postpone a pseudocode of the algorithm of Step 2 to the full version of the paper.

To illustrate the previuos description, let us consider the example of a part of the concept lattice in Fig. 3. The concepts are denoted by circled numbers and the number of lower concepts is written to the right of every concept. We select the tree of concepts as follows. The root node of the tree is the greatest concept 1. As children nodes of the root node are selected concepts 2 and 3 since they form a collection $\mathcal{N}_1^{body\ temp.}$ of all lower neighbor concepts generated by both added (logical) attributes *bt cold* and *bt warm*, respectively, transformed from the selected categorical attribute *body temp.*. Note that we could have as well selected concepts 4 and 5 from the collection $\mathcal{N}_1^{gives\ birth}$, but since both concepts 2 and 4 have the equal minimal number L_2 and L_4 of lower concepts and both cover the equal maximal number of objects/records, we have selected concept 2 and thus the categorical attribute *body temp.* arbitrarily. The edges of the selected tree are labeled by the corresponding logical attributes. Similarly, the children nodes of concept 3 will be concepts 11 and 19, and this is the end of the tree selection step since concepts 4, 11 and 19 have no lower neighbors. The resulting tree of concepts is depicted in Fig. 3 with solid lines.

Let us briefly explain why our selection criterion for the decision attribute is the minimal number of lower concepts of the actual concept (or the maximal number of covered objects, respectively). This criterion simply means the minimal number of added attributes, hence the minimal number of decision attributes in decisions following the actual decision, thus leading to the minimal decision tree (according to the previously mentioned Occam's Razor principle). The point is that the decision minimizing the number of consecutive decisions should be a good decision.

Step 3 The last step (third one) of our method is the transformation of the tree of concepts into a decision tree. A decision tree has in its every node the chosen categorical attribute on which the decision is made and the edges from the node are labeled by the possible values of the attribute. The leaves are labeled by class label(s) of covered records. In the tree of concepts the logical attributes transformed from (and

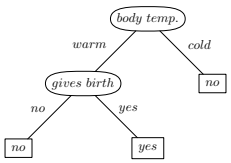

Figure 4: The decision tree of input data in Fig. 2

standing for the values of) the categorical attribute are in the labels of edges connecting concepts. Hence the transformation of the tree of concepts into a decision tree is simple: edges are relabeled to the values of the categorical attribute, inner concepts are labeled by the corresponding categorical attributes, and leaves are labelled by class label(s) of covered objects/records.

The last problem to solve is multiple different class label(s) of covered records in tree leaves. This can happen for several reasons, for example the presence of conflicting records in input data differing in class label(s) only. Common practice for dealing with multiple different target class label(s) is as simple as picking the major class label value(s) as the target classification of records covered by leave node and we adopt this solution. A special case are leave nodes represented by (a copy of) the least concept, since the least concept usually covers no objects/records. These nodes are labelled by the class label(s) of their parent nodes.

The resulting decision tree of input data in Fig. 2 (top) transformed from the tree of concepts in Fig. 3 is depicted in Fig. 4.

4 Comparison with other algorithms

The asymptotic time complexity of the presented algorithm is given by the (part of the) concept lattice building step since this step is the most time demanding. Since the modification of neighbor relation does not alter the asymptotic time complexity, the overall asymptotic complexity of our method is equal to $O(|X||Y|^2|L|)$ in the case of Lindig's NextNeighbor algorithm, for instance. Here, $|X|$ is the number of input records, $|Y|$ is the number of (logical) attributes and $|L|$ is the size of the concept lattice, i.e. the number of all formal concepts.

However, for the decision tree induction problem, accuracy, i.e. the percentage of correctly and incorrectly decided records from both training and testing dataset, is more important than time complexity. We performed preliminary experiments and compared our method to reference decision tree algorithms ID3 and C4.5 (entropy and information gain based) and also to one instance based learning method (IB1) and one artificial neural network trained by back propagation [Mitchel, 1997] (MLP). We implemented our method in C language. All other classifiers were borrowed and run from the Weka[1] (Waikato Environment for Knowledge Analysis), a software package which aids the development of and contains implementations

[1]Weka is a free software available at `http://www.cs.waikato.ac.nz/ml/weka/`

Table 1: Characteristics of datasets used in experiments

Dataset	No. of attributes	No. of records	Class distribution
breast-cancer	6	138	100/38
kr-vs-kp	14	319	168/151
mushroom	10	282	187/95
vote	8	116	54/62
zoo	9	101	41/20/5/13/4/8/10

of several machine learning and data mining algorithms in Java. Default Weka's parameters were used for the algorithms.

The experiments were done on selected public real-world datasets from UCI machine learning repository [Newman, Hettich, Blake and Merz, 1998]. The datasets were cleared of records containing missing values and actually, we selected subcollections of less-valued attributes of each dataset and subcollections of records of some datasets, due to computational time of repeated executions on the same dataset. The basic characteristics of the datasets are depicted in Tab. 1. The results of averaging 10 executions of the 10-Fold Stratified Cross-validation test (which gives total of 100 executions for each algorithm over each dataset) are depicted in Tab. 2. The table shows average percentage rates of correct decisions for both training (upper item in the table cell) and testing (lower item) dataset part, for each compared algorithm and dataset, plus average over all datasets. Bold face numbers denote the best results.

We can see that our FCA based decision tree induction method outperforms all other compared methods on datasets vote and zoo, on both training and testing data, and gains almost identical results to ID3 and MLP on datasets breast-cancer and kr-vs-kp, outperforming C4.5 and IB1. On mushroom dataset, which is quite sparse comparing to the other datasets, our method is little behind ID3, C4.5 and MLP on training data, but, however, almost equal on testing data. Clearly, the FCA based method outperforms instance based learning methods and it seems that it could give better results than traditional decision tree, entropy based, methods and even neural network methods on clear dense data. However, more experiments on additional datasets and deep insight are needed to approve this conclusion. The experiments show that our simple FCA based method is promising in using FCA in the decision tree induction problem.

The bottleneck of the method could be performance, the total time of tree induction, but once one already has the (whole) concept lattice of input data, then the tree selection is very fast. This draws a possible usage and perspective of the method: decision making from already available concept lattices. The advantage of our method over other methods is the conceptual information hidden in tree nodes (note that they are in fact formal concepts). The attributes covered by a node are the attributes common to all objects/records covered by the node, which might be usefull information for furher exploration, application and interpretation of the decision tree. This type of information is not (directly) available by other methods.

Table 2: Percentage correct rates for datasets in Tab. 1

training % *testing %*	breast-cancer	kr-vs-kp	mushroom	vote	zoo	average
FCA based	**88.631**	84.395	96.268	**97.528**	**98.019**	92.968
	79.560	74.656	96.284	90.507	**96.036**	**87.409**
ID3	88.630	**84.674**	**97.517**	97.528	98.019	**93.274**
	75.945	74.503	96.602	89.280	95.036	86.273
C4.5	86.328	82.124	97.163	94.883	96.039	91.307
	79.181	72.780	**96.671**	86.500	92.690	85.564
IB1	84.887	79.132	96.556	97.020	97.799	91.079
	71.901	68.886	95.214	**91.303**	94.463	84.353
MLP	88.550	84.426	97.234	95.545	97.678	92.687
	79.939	**74.880**	95.992	88.106	95.536	86.891

5 Conclusion and topics for future research

We have presented a simple novel method of decision tree induction by selection of the tree of concepts from a concept lattice. The criterion of choosing an attribute based on which the node of the tree is split is determined by the number of all lower concepts of the concept corresponding to the node. The approach interconnects areas of decision trees and formal concept analysis. We have also presented some comparison to classical decision tree algorithms, namely ID3 and C4.5, and also to instance based learning and neural network methods, and have seen that our method compares quite well and surely deserves more attention. Topics for future research include:

- Explore the possibility to compute a smaller number of formal concepts from which the nodes of a decision tree is constructed.
- The problems of overfitting [Mitchel, 1997] in data and uncomplete data, i.e. data having missing values for some attributes in some records.
- Incremental updating of induced decision trees via incremental methods of building concept lattices.

References

[Belohlavek, De Baets, Outrata and Vychodil, 2007] R. Belohlavek, B. De Baets, J. Outrata, V. Vychodil. Trees in Concept Lattices. In: V. Torra, Y. Narukawa, Y. Yoshida (Eds.): Modeling Decisions for Artificial Intelligence: 4th International Conference, MDAI 2007, *Lecture Notes in Artificial Intelligence* **4617**, pp. 174–184, Springer-Verlag, Berlin/Heidelberg, 2007.

[Belohlavek and Sklenar, 2005] R. Belohlavek, V. Sklenar. Formal concept analysis constrained by attribute-dependency formulas. In: B. Ganter and R. Godin (Eds.): ICFCA 2005, *Lecture Notes in Computer Science* **3403**, pp. 176–191, Springer-Verlag, Berlin/Heidelberg, 2005.

[Carpineto and Romano, 2004] C. Carpineto, G. Romano. *Concept Data Analysis. Theory and Applications.* J. Wiley, 2004.

[Dunham, 2003] M. H. Dunham. *Data Mining. Introductory and Advanced Topics.* Prentice Hall, Upper Saddle River, NJ, 2003.

[Fu, Fu, Njiwoua and Mephu Nguifo, 2004] H. Fu, H. Fu, P. Njiwoua, E. Mephu Nguifo. A Comparative Study of FCA-Based Supervised Classification Algorithms. 2nd Intl. Conf. on Formal Concept Analysis (ICFCA), *LNAI* **2961**, pp. 313–320, Springer-Verlag, 2004.

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

[Kuznetsov, 2004] S. O. Kuznetsov. Machine Learning and formal Concept Analysis. In P. Eklund (Ed.): Concept Lattices: 2nd Int. Conf. on Formal Concept Analysis (ICFCA), *LNAI* **2961**, pp. 287–312, Springer-Verlag, 2004.

[Mephu Nguifo and Njiwoua, 2001] E. Mephu Nguifo, P. Njiwoua. IGLUE: A lattice-based constructive induction system. *Intell. Data Anal.* **5**(1), pp. 73–91, 2001.

[Newman, Hettich, Blake and Merz, 1998] D. J. Newman, S. Hettich, C. L. Blake, C. J. Merz. *UCI Repository of machine learning databases*, [`http://www.ics.uci.edu/~mlearn/MLRepository.html`]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[Lindig, 2000] C. Lindig. Fast concept analysis. In: Stumme G.: *Working with Conceptual Structures – Contributions to ICCS 2000.* Shaker Verlag, Aachen, 2000, 152–161.

[Mingers, 1987] J. Mingers. Expert systems – rule induction with statistical data. *J. of the Operational Research Society.*, **38**, pp. 39–47, 1987.

[Mitchel, 1997] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[Pistori and Neto, 2003] H. Pistori, J. J. Neto. Decision Tree Induction using Adaptive FSA. *CLEI Electron. J.* **6**(1), 2003.

[Quinlan, 1993] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Quinlan, 1996] J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys*, **28**(1), 1996.

[Tan, Steinbach and Kumar, 2006] P. N. Tan, M. Steinbach, V. Kumar. *Introduction to Data Mining.* Addison Wesley, Boston, MA, 2006.

International Journal of General Systems
Vol. 38, No. 4, May 2009, 455–467

Taylor & Francis
Taylor & Francis Group

Inducing decision trees via concept lattices[1]

Radim Belohlavek[ac], Bernard De Baets[b], Jan Outrata[c]* and Vilem Vychodil[ac]

[a]*Department of Systems Science and Industrial Engineering, T. J. Watson School of Engineering and Applied Science, Binghamton University – SUNY, Binghamton, NY 13902, USA;* [b]*Department of Applied Mathematics, Biometrics, and Process Control, Ghent University, Coupure links 653, B-9000 Gent, Belgium;* [c]*Department of Computer Science, Palacky University Olomouc, Tomkova 40, CZ-779 00 Olomouc, Czech Republic*

(*Received 21 December 2007; final version received 7 October 2008*)

We present a novel method for the construction of decision trees. The method utilises concept lattices in that certain formal concepts of the concept lattice associated to input data are used as nodes of the decision tree constructed from the data. The concept lattice provides global information about natural clusters in the input data, which we use for selection of splitting attributes. The usage of such global information is the main novelty of our approach. Experimental evaluation indicates good performance of our method. We describe the method, experimental results, and a comparison with standard methods on benchmark datasets.

Keywords: decision trees; classification; machine learning; concept lattice; formal concept analysis

1. Introduction

Decision trees represent the most commonly used method in data mining and machine learning (Quinlan 1993, Dunham 2003, Tan *et al.* 2006). A decision tree is typically used for a classification of objects into a given set of classes based on the objects' attributes. Many algorithms for the construction of decision trees are proposed in the literature, see e.g. (Tan *et al.* 2006).

This paper presents a novel approach to decision tree construction, which is based on formal concept analysis (FCA) of the input data. Our approach utilises concept lattices in that certain formal concepts associated to input data are used as nodes of the decision tree constructed from the input data. The concept lattice provides global information about natural clusters, represented by formal concepts, in the input data. Using formal concepts as nodes of decision trees is a straightforward idea because both formal concepts and decision tree nodes represent collections of records (objects) in the input data defined by having the same values for certain attributes. A challenge exists in how to select good formal concepts for decision tree nodes. We attempt to consider a concept lattice (without the least formal concept) as a collection of overlapping trees. The construction of a decision tree is then reduced to the problem of selection of one of these trees.

FCA and concept lattices are utilised in several machine learning and decision tree induction algorithms proposed in the literature. For instance, Carpineto and Romano (1996) present GALOIS, a clustering method based on concept lattices, in which similarity

*Corresponding author. Email: jan.outrata@upol.cz

ISSN 0308-1079 print/ISSN 1563-5104 online

DOI: 10.1080/03081070902857563
http://www.informaworld.com

between objects and clusters is defined as the number of common attributes shared by objects. In (Mephu Nguifo and Njiwoua 2001), the authors use FCA in IGLUE, a method which selects relevant categorical attributes and transforms them into continuous numerical attributes which are then used to solve a decision problem by k-nearest neighbour clustering. Another approach utilising FCA is described in Kuznetsov (2004) which presents a model of learning from positive and negative examples. Fu *et al.* (2004) provides a survey and theoretical and experimental comparison of several FCA-based classification algorithms. Note that FCA-based approaches are commonly called lattice-based or concept-based learning techniques in data mining (Fayyad *et al.* 1996, Pasquier *et al.* 1999).

The paper is organised as follows. The next section contains preliminaries from decision trees and formal concept analysis. In Section 3 we present our approach including an algorithm for inducing decision trees. The algorithm is accompanied by an illustrative example. An experimental evaluation of our method and a comparison to standard methods on benchmark datasets is provided in Section 4. Section 5 presents conclusions and outlines topics for future research.

2. Preliminaries

2.1 *Decision trees*

A decision tree can be considered as a tree representation of a function over attributes which takes a finite number of values. The goal is to construct a tree that approximates a given function, partially described by a table containing records in its rows, with a desired accuracy. Every record consists of particular values of the function input values (attribute values) and the corresponding output value (class label). For an object described by its attribute values, the value assigned by the decision tree to those attribute values is considered the label of a class to which the object belongs. A good decision tree is supposed to classify well both the data described by the table records as well as 'unseen' data.

Each non-leaf node of a decision tree is labelled by an attribute, called a splitting attribute for this node. Such a node represents a test, according to which records are split into n classes which correspond to n possible outcomes of the test. In the basic setting, the outcomes are represented by values of the splitting attribute. Leaf nodes of the tree represent collections of records all of which, or the majority of which, have the same function value (class label). For example, the table in Figure 1 (top) describes a partial function $f : A \times B \times C \to D$. The decision trees in Figure 1 (bottom) represent two functions, both of which are extensions of f.

A strategy commonly used in the existing algorithms for inducing decision trees from data consists of constructing a decision tree in a top-down fashion, from the root node to the leaves, by successively splitting existing nodes and creating new ones. For every node, a splitting attribute is chosen to split the collection of records covered by the node into smaller collections, which correspond to values of the splitting attribute. For every such value, a new node is attached as a child to the node for which the splitting attribute has been chosen. The process continues recursively until all the records corresponding to any leaf node, or a prescribed majority of them, belong to the same class. A critical point in this strategy is the selection of splitting attributes, for which many approaches were proposed. These include the well-known approaches based on entropy measures, Gini index, classification error, or other measures defined in terms of class distribution of the records before and after splitting (see Quinlan 1996, Murthy 1998, Tan *et al.* 2006 for overviews).

A	B	C	f(A,B,C)
good	yes	false	yes
good	no	false	no
bad	no	false	no
good	no	true	yes
bad	yes	true	yes

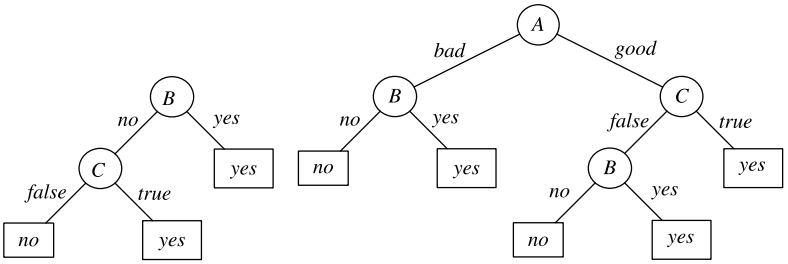

Figure 1. Two decision trees (bottom) representing functions which extend the partial function f (top).

2.2 *Formal concept analysis*

FCA is a method for analysis of object-attribute data (Ganter and Wille 1999, Carpineto and Romano 2004). Such data is usually described by a table with rows and columns representing objects and attributes, respectively, and with table entries containing attribute values, which the objects have. In the basic setting, FCA deals with binary attributes, i.e. every attribute applies or does not apply to a particular object. Many-valued attributes, such as nominal and ordinal attributes, are transformed to binary ones using so-called conceptual scaling. FCA produces two kinds of output from a given dataset. The first output is called a concept lattice. A concept lattice is a partially ordered collection of particular clusters called formal concepts. The second output consists of a non-redundant base of particular attribute dependencies called attribute implications.

We now summarise basic notions of FCA. An object-attribute data table can be identified with a triplet $\langle X, Y, I\rangle$ where X is a non-empty set of objects, Y is a non-empty set of attributes, and $I \subseteq X \times Y$ is an object-attribute relation. Objects and attributes correspond to table rows and columns, respectively, and $\langle x, y\rangle \in I$ indicates that object x has attribute y (table entry corresponding to row x and column y contains $\times$; if $\langle x, y\rangle \notin I$ the table entry contains blank symbol). In terms of FCA, $\langle X, Y, I\rangle$ is called a formal context. For every $A \subseteq X$ and $B \subseteq Y$ denote by $A^{\uparrow}$ a subset of Y and by $B^{\downarrow}$ a subset of X defined as

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

That is, $A^{\uparrow}$ is the set of all attributes from Y shared by all objects from A (and similarly for $B^{\downarrow}$). A formal concept in $\langle X, Y, I\rangle$ is a pair $\langle A, B\rangle$ of $A \subseteq X$ and $B \subseteq Y$ satisfying $A^{\uparrow} = B$ and $B^{\downarrow} = A$. That is, a formal concept consists of a set A (so-called extent) of objects which are covered by the concept and a set B (so-called intent) of attributes which are covered by the concept such that A is the set of all objects sharing all attributes from B and, conversely, B is the collection of all attributes from Y shared by all objects from A. Alternatively, formal concepts can be defined as maximal rectangles of $\langle X, Y, I\rangle$ which are full of $\times$'s: For $A \subseteq X$ and $B \subseteq Y$, $\langle A, B\rangle$ is a formal concept in $\langle X, Y, I\rangle$ iff $A \times B \subseteq I$

and there is no $A' \supset A$ or $B' \supset B$ such that $A' \times B \subseteq I$ or $A \times B' \subseteq I$. Formal concepts represent clusters hidden in object-attribute data.

A set $\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \,|\, A^{\uparrow} = B, B^{\downarrow} = A\}$ of all formal concepts in $\langle X, Y, I\rangle$ can be equipped with a partial order $\leq$. The partial order models a subconcept–superconcept hierarchy, e.g. *dog* $\leq$ *mammal*, and is defined by

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \quad \text{iff } A_1 \subseteq A_2 (\text{iff } B_2 \subseteq B_1). \tag{1}$$

Note that $^{\uparrow}$ and $^{\downarrow}$ form a Galois connection (Ganter and Wille 1999) and that $\mathcal{B}(X, Y, I)$ is in fact a set of all fixpoints of $^{\uparrow}$ and $^{\downarrow}$. $\mathcal{B}(X, Y, I)$ equipped with $\leq$ happens to be a complete lattice, called the concept lattice of $\langle X, Y, I\rangle$. The basic structure of concept lattices is described by the so-called basic theorem of concept lattices (Ganter and Wille 1999).

THEOREM 2.1. (1) The set $\mathcal{B}(X, Y, I)$ equipped with $\leq$ forms a complete lattice in which infima and suprema are given by

$$\bigwedge_{j\in J} \langle A_j, B_j\rangle = \left\langle \bigcap_{j\in J} A_j, \left(\bigcup_{j\in J} B_j\right)^{\downarrow\uparrow} \right\rangle, \quad \bigvee_{j\in J} \langle A_j, B_j\rangle = \left\langle \left(\bigcup_{j\in J} A_j\right)^{\uparrow\downarrow}, \bigcap_{j\in J} B_j \right\rangle.$$

(2) Moreover, an arbitrary complete lattice $\mathbf{V} = \langle V, \leq\rangle$ is isomorphic to $\mathcal{B}(X, Y, I)$ iff there are mappings $\gamma : X \to V$, $\mu : Y \to V$ such that

(i) $\gamma(X)$ is $\bigvee$-dense in V, $\mu(Y)$ is $\bigwedge$-dense in V;
(ii) $\gamma(x) \leq \mu(y)$ iff $\langle x, y\rangle \in I$.

Recall that the cover relation on $\mathcal{B}(X, Y, I)$ is defined as follows. A formal concept $\langle A_1, B_1\rangle$ covers a formal concept $\langle A_2, B_2\rangle$ if $\langle A_2, B_2\rangle \leq \langle A_1, B_1\rangle$ and there is no $\langle A_3, B_3\rangle$ distinct from both $\langle A_1, B_1\rangle$ and $\langle A_2, B_2\rangle$ such that $\langle A_2, B_2\rangle \leq \langle A_3, B_3\rangle \leq \langle A_1, B_1\rangle$.

For detailed information on formal concept analysis we refer to Carpineto and Romano (2004) and Ganter and Wille (1999) where the reader can find theoretical foundations, methods and algorithms, and applications in various areas.

3. Decision tree induction based on FCA

In this section, we describe our algorithm for the induction of decision trees. As mentioned above, the algorithm utilises a concept lattice associated to input data, i.e. a partially ordered set of formal concepts in the sense of FCA. In particular, we attempt to consider the concept lattice (without the least formal concept) as a collection of overlapping trees and to select one tree from this collection as the resulting decision tree.

Input data and its transformation. We consider input data with categorical attributes in our paper. To derive a concept lattice from the input data, we need to transform the categorical attributes to binary attributes because, in its basic setting, FCA works with binary attributes. A transformation of input data, which consists in replacing non-binary attributes into binary ones is called conceptual scaling in FCA (Ganter and Wille 1999). Note that in our case, we need not transform the class label attribute, i.e. the attribute determining to which class the record belongs, because we build the concept lattice over the input attributes only. Throughout this section, we use the input data from Table 1 (top) to illustrate the main issues involved. The data table contains sample animals described by attributes *body temperature*, *gives birth*,

Table 1. Input data table (top) and corresponding data table for FCA (bottom).

Name	*body temp.*	*gives birth*	*four-legged*	*hibernates*	*mammal*
cat	warm	yes	yes	no	yes
bat	warm	yes	no	yes	yes
salamander	cold	no	yes	yes	no
eagle	warm	no	no	no	no
guppy	cold	yes	no	no	no

Name	*bt cold*	*bt warm*	*gb no*	*gb yes*	*fl no*	*fl yes*	*hb no*	*hb yes*	*mammal*
cat	0	1	0	1	0	1	1	0	yes
bat	0	1	0	1	1	0	0	1	yes
salamander	1	0	1	0	0	1	0	1	no
eagle	0	1	1	0	1	0	1	0	no
guppy	1	0	0	1	1	0	1	0	no

four-legged, *hibernates*, and *mammal*, with the last attribute being the class label attribute. After an obvious transformation (nominal scaling) of the input attributes, we obtain the data depicted in Table 1 (bottom). A concept lattice which we use in our method is derived from data which we obtain after such transformation.

Next, we describe our algorithm. Step 1 describes how we compute the (part of) concept lattice of the input data. Step 2 describes the selection of a tree from the concept lattice computed in Step 1. Step 3 describes how we build the decision tree from the tree computed in Step 2.

Step 1. In this step, we compute a part of the concept lattice associated to the data corresponding to input attributes. For this purpose, we use the well-known Lindig's algorithm (Lindig 2000), which we modify in two respects. The original Lindig's algorithm, in its top-down version, generates all formal concepts of a concept lattice associated to input data and the cover relation. It starts with the largest formal concept and recursively generates all formal concepts that are covered by largest formal concept, then generates all formal concepts which are covered by those covered by the largest formal concept, and so on until all formal concepts have been computed. Our modification of Lindig's algorithm consists of two steps.

First, we do not generate lower neighbors of formal concept whose extent contains objects that have all the same class label. That is, if c is the class label attribute, we do not generate lower neighbors of formal concepts $\langle A, B\rangle$ such that for every $x_1, x_2 \in A$, the value of c on x_1 equals the value of c on x_2.

Second, contrary to Lindig's algorithm which computes all formal concepts and the cover relation, our modification computes a relation on the set of the computed formal concepts which is in general larger than the cover relation. In the original Lindig's algorithm, procedure NEXTNEIGHBOR generates set *(Next)Neighbors* of $\langle A, B\rangle$ defined by

$$(Next)Neighbors \text{ of } \langle A, B\rangle = \{\langle C, D\rangle \mid D = (B \cup \{y\})^{\downarrow\uparrow},$$

$$y \in Y - B \text{ such that } (B \cup \{z\})^{\downarrow\uparrow} = D \text{ for all } z \in D - B\}.$$

It can be shown that *(Next)Neighbors* of $\langle A, B\rangle$ is just the set of all formal concepts covered by $\langle A, B\rangle$. In our modification, the procedure NEXTNEIGHBOR generates the set

(*Our*)*Neighbors* of $\langle A, B\rangle$ defined by

$$(Our)Neighbors \text{ of } \langle A, B\rangle = \{\langle C, D\rangle \mid D = (B \cup \{y\})^{\downarrow\uparrow},\ y \in Y - B\}.$$

Clearly, (*Our*)*Neighbors* of $\langle A, B\rangle$ is (in general) larger than (*Next*)*Neighbors* of $\langle A, B\rangle$ because it may contain formal concepts which result by adding a single attribute $y \in Y - B$ but are not covered by $\langle A, B\rangle$.

The reason for our modification is the following. As mentioned above, formal concepts correspond to the nodes of a decision tree in our approach. Let a formal concept $\langle A, B\rangle$ correspond to a node n in a decision tree. Let $y \in Y - B$ be a binary attribute corresponding to value v_y of a categorical attribute a_y. That is, a_y has value v_y for object x in the original input data if and only if y has value 1 for x in the transformed data with binary attributes. If a_y is the splitting attribute for node n, then $\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\rangle$ is the formal concept corresponding to node n_y which is connected to n via an edge representing the test 'is the value of a_y equal to v_y?' In order to keep the possibility of having nodes n and n_y in the resulting decision tree, we need to generate both $\langle A, B\rangle$ and $\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\rangle$ even if $\langle (B \cup \{y\})^{\downarrow}, (B \cup \{y\})^{\downarrow\uparrow}\rangle$ is not covered by $\langle A, B\rangle$ in the concept lattice. This is why (*Our*)*Neighbors* of $\langle A, B\rangle$ is in general different from (*Next*)*Neighbors* of $\langle A, B\rangle$.

Algorithm 1 contains pseudocode of the modified Lindig's algorithm. NEXTNEIGHBOR is the main procedure. Formal concepts computed by the algorithm are stored in the variable $\mathcal{F}$. Variable $\langle A, B\rangle_*$ stores the lower neighbours of $\langle A, B\rangle$. The procedure DECIDED prevents computing lower neighbours of formal concepts whose objects have the same class label. Procedure NEIGHBORS computes the set (*Our*)*Neighbors* of $\langle A, B\rangle$. The algorithm also computes for every formal concept $\langle A, B\rangle$ the number $L_{\langle A,B\rangle}$ explained and utilised in Step 2 below. The part of the concept lattice built from the data table in Table 1 (bottom) computed by Algorithm 1 is shown in Figure 2. Note that the new lower neighbour relationship is displayed by dashed lines. The solid lines indicate a tree to be selected from the part of the concept lattice using a procedure described in Step 2.

Step 2. In this step, we select a tree of formal concepts from the part of a concept lattice computed in Step 1. For this purpose, we compute for every formal concept $\langle A, B\rangle$ computed in Step 1 the number $L_{\langle A,B\rangle}$ of all formal concepts which can be reached from $\langle A, B\rangle$ by following certain labelled paths from $\langle A, B\rangle$ downward. As mentioned in Step 1, the numbers $L_{\langle A,B\rangle}$ are computed in Algorithm 1. The paths consist of labelled edges corresponding to the lower neighbour relation described by (*Our*)*Neighbors* of .... In particular, an edge

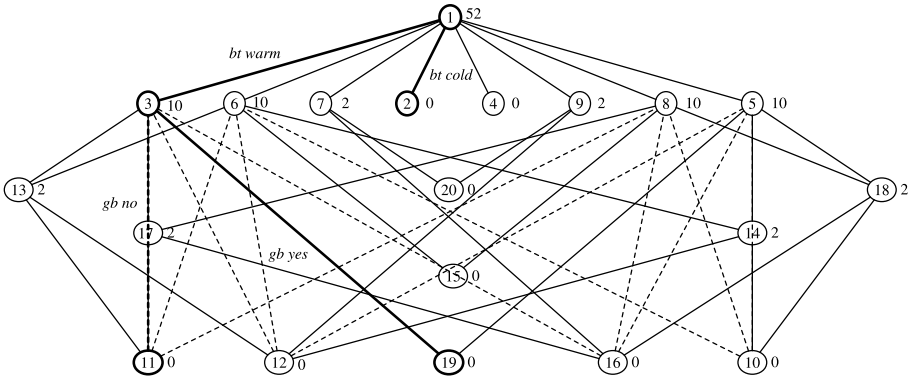

Figure 2. Part of the concept lattice and tree of concepts (solid lines) of data table in Table 1.

with label y goes from $\langle C_1, D_1\rangle$ to $\langle C_2, D_2\rangle$ if $D_2 = (D_1 \cup \{y\})^{\downarrow\uparrow}$, i.e. if $\langle C_2, D_2\rangle$ belongs to (*Our*)*Neighbors* of $\langle C_1, D_1\rangle$ due to attribute y. Therefore, $L_{\langle A,B\rangle}$ is the number of formal concepts which can be reached from $\langle A, B\rangle$ via the lower neighbour relation in a way that counts a lower neighbour $\langle C_2, D_2\rangle$ of $\langle C_1, D_1\rangle$ multiple times. Namely, $\langle C_2, D_2\rangle$ is counted k times where k is the number of attributes due to which $\langle C_2, D_2\rangle \in$ (*Our*)*Neighbors* of $\langle C_1, D_1\rangle$, i.e. $k = |\{y; D_2 = (D_1 \cup \{y\})^{\downarrow\uparrow}\}|$. The multiple counting of formal concepts is ensured by the labelling used in $\langle\langle C, D\rangle, y\rangle$ in Algorithm 1.

Furthermore, for every formal concept $\langle A, B\rangle$ we define collections $\mathcal{N}^a_{\langle A,B\rangle}$ of formal concepts that are candidates to become the children of $\langle A, B\rangle$ in the selected tree. a denotes a categorical attribute from the original input data. Note that every binary attribute y_v from the data obtained by transformation from the original input data corresponds to some value v of some categorical attribute a. $\mathcal{N}^a_{\langle A,B\rangle}$ is the collection of lower neighbour concepts of $\langle A, B\rangle$ defined as follows:

(a) for every value v of the categorical attribute a and the corresponding binary attribute y_v, if the formal concept $\langle C, D\rangle$ from (*Our*)*Neighbors* of $\langle A, B\rangle$ which results by adding attribute y_v belongs to the part of the concept lattice computed in Step 1, then $\langle C, D\rangle$ belongs to $\mathcal{N}^a_{\langle A,B\rangle}$;

(b) if some formal concept $\langle C, D\rangle$ from (a) does not belong to the part of the concept lattice computed in Step 1, $\mathcal{N}^a_{\langle A,B\rangle}$ contains the least formal concept $\langle Y^{\downarrow}, Y\rangle$ (in this case, the least formal concept 'replaces'' $\langle C, D\rangle$ in $\mathcal{N}^a_{\langle A,B\rangle}$), and we put $L_{\langle Y^{\downarrow},Y\rangle} = \infty$.

Algorithm 1 Computing part of concept lattice

```
procedure NEIGHBORS(⟨A, B⟩):
    L := ∅
    for each y ∈ Y \ B:
      C := (B ∪ {y})↓
      D := C↑
      if D ≠ Y:
        add ⟨⟨C, D⟩, y⟩ to L
    return L

procedure DECIDED(⟨A, B⟩):
  if all x ∈ A have the same class label
    return true
  else
    return false

procedure GENERATEFROM(⟨A, B⟩):
  ⟨A, B⟩* := call NEIGHBORS(⟨A, B⟩)
  N := {⟨C, D⟩ | ⟨⟨C, D⟩, y⟩ ∈ ⟨A, B⟩* and
    ⟨C, D⟩ ∉ F}
  for each ⟨C, D⟩ ∈ N:
    add ⟨C, D⟩ to F
    L⟨C,D⟩ := 0
  for each ⟨C, D⟩ ∈ N:
    if not call DECIDED(⟨C, D⟩):
      call GENERATEFROM(⟨C, D⟩)
  for each ⟨⟨C, D⟩, y⟩ ∈ ⟨A, B⟩*:
    add 1 + L⟨C,D⟩ to L⟨A,B⟩

procedure NEXTNEIGHBOR:
  F := ∅
  add ⟨∅↓, ∅↓↑⟩ to F
  L⟨∅↓,∅↓↑⟩ := 0
  if not call DECIDED(⟨∅↓, ∅↓↑⟩):
    call GENERATEFROM(⟨∅↓, ∅↓↑⟩)
  return P := ⟨F, {⟨A, B⟩* | ⟨A, B⟩ ∈ F}⟩
```

Next, we select a tree from the part of the concept lattice computed in Step 1 by iteratively going from the largest formal concept to the minimal ones. The selection is based on the numbers $L_{\langle A,B\rangle}$ defined above.

(1) The root node of the tree is the largest formal concept $\langle X, X^{\uparrow}\rangle$.
(2) This step corresponds to selection of the splitting attribute. For every formal concept $\langle A, B\rangle$ in the tree which we construct we select from among all the categorical attributes a categorical attribute a for which

$$\min\left\{L_{\langle C,D\rangle} \mid \langle C, D\rangle \in \mathcal{N}^{a}_{\langle A,B\rangle}\right\}$$

attains the minimum value, i.e. we select a for which $\mathcal{N}^{a}_{\langle A,B\rangle}$ contains a formal concept $\langle C, D\rangle$ with the smallest number $L_{\langle C,D\rangle}$. The idea behind this rule is that a small value of $L_{\langle C,D\rangle}$ indicates, in the optimistic scenario, a small number of decision steps necessary to classify objects from A provided we start with a decision based on a, because there is a short classification path in the decision tree going from the node corresponding to $\langle A, B\rangle$. In case of a tie, i.e. if $L_{\langle C_1,D_1\rangle} = L_{\langle C_2,D_2\rangle}$ for some $a_1 \neq a_2$ with $\langle C_1, D_1\rangle \in \mathcal{N}^{a_1}_{\langle A,B\rangle}$ and $\langle C_2, D_2\rangle \in \mathcal{N}^{a_2}_{\langle A,B\rangle}$, we select a_i for which the extent C_i is the largest. If there is still a tie, we break it arbitrarily. The resulting categorical attribute a is later used as the splitting attribute for the node of the decision tree that corresponds to formal concept $\langle A, B\rangle$.
(3) For every formal concept $\langle A, B\rangle$ in the tree and the categorical attribute a selected for $\langle A, B\rangle$ in (2) we connect $\langle A, B\rangle$ to each formal concept $\langle C, D\rangle$ from $\mathcal{N}^{a}_{\langle A,B\rangle}$ by an edge labelled by a binary attribute y for which $D = (B \cup \{y\})^{\downarrow\uparrow}$.

Algorithm 2 contains a pseudocode of the algorithm that selects a tree from the part of the concept lattice. SELECTTREE is the main procedure. Formal concepts of the selected tree are stored in variable $\mathcal{G}$. Edges between nodes are represented by variables $\langle A, B\rangle_{+}$. The procedure CHILDREN calculates $\mathcal{N}^{a}_{c}$.

To illustrate the previous description, consider the part of the concept lattice presented in Figure 2. Formal concepts of this part are represented by circles, which contain the numbers from 1 to 20 assigned to the formal concepts. For every formal concept $\langle A, B\rangle$, the number $L_{\langle A,B\rangle}$ is attached to the right of the circle representing $\langle A, B\rangle$. Algorithm 2 selects a tree of formal concepts as follows. The root node of the tree is the formal concept No. 1. Formal concepts No. 2 and 3 are then selected as the children of the root since they are the elements of the set $\mathcal{N}^{\text{body temp.}}_{1}$ of lower nodes corresponding to the attribute body temp. which satisfies the conditions described in (2) above. Note that in this case, we could have chosen $\mathcal{N}^{\text{gives birth}}_{1}$ instead of $\mathcal{N}^{\text{body temp.}}_{1}$, since both the formal concept No. 2 from $\mathcal{N}^{\text{body temp.}}_{1}$ and No. 4 from $\mathcal{N}^{\text{gives birth}}_{1}$ have the same minimal number L_2 and L_4 and both contain the same number of objects in their extents. The edges of the selected tree are labelled by binary attributes as described in (3) above. Similarly, the children of the formal concept No. 3 are the formal concepts No. 11 and No. 19. This ends the tree selection because the formal concepts No. 4, No. 11, and No. 19 have no lower neighbours. The resulting tree is depicted in Figure 2 by the solid lines.

Step 3. In this step, the tree obtained in Step 2 is transformed into a decision tree. This step is straightforward. We take the tree obtained in Step 2 and re-label its nodes and edges. An inner node is labelled by the categorical attribute selected in (2) of Step 2 for this node. For example, when constructing the decision tree from the tree selected in Figure 2, the node corresponding to the formal concept No. 3 is labelled by gives birth. An edge going from a node is labelled by the value of a categorical attribute corresponding to the

Algorithm 2 Selection of tree of formal concepts

procedure CHILDREN ($\langle\langle A, B\rangle, y\rangle$):
  $a :=$ *categorical attribute for* y
  $\mathcal{N}^a_{\langle A,B\rangle} := \emptyset$
  for each *logical attribute* z *for* a:
    search for $\langle\langle C, D\rangle, x\rangle \in \langle A, B\rangle_*$ ***such that*** $x = z$
    if found:
      add $\langle\langle C, D\rangle, z\rangle$ ***to*** $\mathcal{N}^a_{\langle A,B\rangle}$
    else:
      add $\langle\langle Y^{\downarrow}, Y\rangle, z\rangle$ ***to*** $\mathcal{N}^a_{\langle A,B\rangle}$
  return $\mathcal{N}^a_{\langle A,B\rangle}$

procedure SELECTFROM ($\langle A, B\rangle$):
  $S := \{y \mid \langle\langle C, D\rangle, y\rangle \in \langle A, B\rangle_* \wedge L_{\langle C,D\rangle}$ *is minimal*$\}$
  if $S = \emptyset$:
    return
  if $|S| \geq 1$ ***and*** *all* $y \in S$ *are for different categorical attribute* a:
    $S := \{y \mid y \in S \wedge \langle\langle C, D\rangle, y\rangle \in \langle A, B\rangle_* \wedge |C|$ *is maximal*$\}$
  select *arbitrary* y ***from*** S
  $\langle A, B\rangle_+ :=$ ***call*** CHILDREN ($\langle\langle A, B\rangle, y\rangle$)
  for each $\langle\langle C, D\rangle, y\rangle \in \langle A, B\rangle_+$:
    add $\langle C, D\rangle$ ***to*** $\mathcal{G}$
    call SELECTFROM ($\langle C, D\rangle$)

procedure SELECTTREE ($\mathcal{P}$):
  $\mathcal{G} := \emptyset$
  add $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$ ***to*** $\mathcal{G}$
  $L_{\langle Y^{\downarrow},Y\rangle} := \infty$
  call SELECTFROM ($\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$)
  return $\langle \mathcal{G}, \{\langle A, B\rangle_+ | \langle A, B\rangle \in \mathcal{G}\}\rangle$

binary attribute used as a label of this edge in (3) of Step 3. For example, the edge labelled by *gb no* in Figure 2 is labelled by *no* in the resulting decision tree.

The last problem is the labelling of leaf nodes. Consider a leaf node of a tree obtained in Step 2 corresponding to formal concept $\langle A, B\rangle$. If all objects from A have the same class label c, or if c is the class label of a majority of objects from A, the corresponding node of the decision tree is labelled by c. If a leaf node n of a tree obtained in Step 2 corresponds to the least formal concept $\langle Y^{\downarrow}, Y\rangle$, cf. (b) of Step 2, the corresponding node of the decision tree is labelled by the label which would have been assigned to a leaf node corresponding to the formal concept of the parent node of n.

The resulting decision tree of the input data in Table 1 (top) which result by the transformation of the tree of formal concepts displayed in Figure 2, as described in Step 3, is depicted in Figure 3.

4. Experimental evaluation

In this section, we describe experiments with our algorithm and its comparison to reference algorithms for decision tree induction. Namely, we compared our algorithm with decision tree algorithms ID3 and C4.5 (entropy and information gain based), an instance based learning method (IB1), and a multilayer perceptron (MLP) neural network trained by back propagation (Mitchell (1997)). We implemented our method in the C language. The other algorithms were borrowed and run from Weka[2] (Waikato Environment for Knowledge Analysis; Witten and Frank 2005), a software package that contains implementations of machine learning and data mining algorithms in Java. Default Weka's

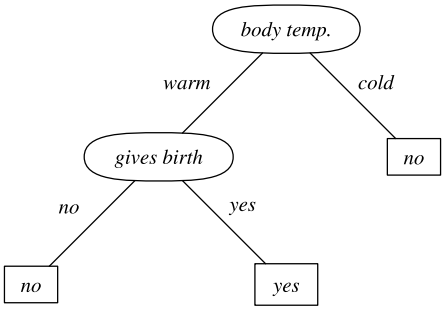

Figure 3. The decision tree of the input data in Table 1.

parameters were used for the other algorithms and decision tree pruning was turned off where available.

The experiments were done on selected public real-world datasets from UCI Machine Learning Repository (Newman *et al.* 1998). The selected datasets are from different areas (medicine, biology, zoology, politics, games). All the datasets contain only categorical attributes with one class label attribute. Due to computational time demands, the datasets were cleared of records containing missing values, of attributes with large numbers of attribute values, and of randomly selected records, to allow for repeated executions of the methods. Let us note in this context that our method is computationally more demanding than the other methods, because of the need to compute a possibly large number of formal concepts. Basic characteristics of the datasets are depicted in Table 2. For datasets with already defined training and testing set (spect and monks-problems) the upper numbers in the table cells relate to the training set and the lower numbers to the testing set. For datasets without training and testing sets, the experiments were done using the 10-fold stratified cross-validation test. The results of averaging 10 execution runs on each dataset with randomly ordered records are depicted in Table 3. The table shows the average percentage rates of correct classifications for both training (upper number in the table cell) and testing (lower number) datasets for each algorithm and

Table 2. Characteristics of datasets used in experiments.

Dataset $\begin{pmatrix}\text{training}\\ \text{testing}\end{pmatrix}$	*No. of attributes*	*No. of records*	*Class distribution*
breast-cancer	6	138	100/38
kr-vs-kp	14	319	168/151
mushroom	10	282	187/95
tic-tac-toe	27	239	156/83
vote	8	116	54/62
zoo	9	101	41/20/5/13/4/8/10
spect	22	80	40/10
		187	15/172
monks-problems-1	17	124	62/62
		432	216/216
monks-problems-2	17	169	105/64
		432	290/142
monks-problems-3	17	122	62/60
		432	204/228

Table 3. Classification accuracy for datasets from Table 2.

Training % *Testing %*	*FCA based*	*ID3*	*C4.5*	*IB1*	*MLP*
breast-cancer	**88.631**	88.630	86.328	84.887	88.550
	79.560	75.945	79.181	71.901	**79.939**
kr-vs-kp	84.395	**84.674**	82.124	79.132	84.426
	74.656	74.503	72.780	68.886	**74.880**
mushroom	96.268	**97.517**	97.163	96.556	97.234
	96.284	96.602	**96.671**	95.214	95.992
tic-tac-toe	98.991	**100.000**	95.165	**100.000**	**100.000**
	85.197	80.519	78.539	83.262	**97.827**
vote	**97.528**	**97.528**	94.883	97.020	95.545
	90.507	89.280	86.500	**91.303**	88.106
zoo	**98.019**	**98.019**	96.039	97.799	97.678
	96.036	95.036	92.690	94.463	95.536
spect	**92.250**	**92.250**	89.250	88.250	91.500
	55.187	54.866	59.679	59.251	**60.481**
monks-problems-1	**100.000**	**100.000**	96.532	**100.000**	99.193
	85.648	79.259	76.828	74.722	**95.833**
monks-problems-2	**100.000**	99.763	92.958	**100.000**	**100.000**
	63.518	59.976	62.314	68.055	**99.814**
monks-problems-3	**100.000**	**100.000**	98.360	**100.000**	99.016
	90.694	91.041	**92.870**	78.634	**92.870**
average	95.608	**95.838**	92.880	94.364	95.314
	81.729	79.703	79.805	78.569	**88.345**

Best performance is in bold.

dataset being compared, plus the average over all datasets. Boldface numbers denote the best results.

We can see that our method, which we call FCA based, outperforms C4.5 and IB1 and gains almost identical results to ID3 and MLP on the training sets of all datasets. On the testing sets this is also the case with the exception of tic-tac-toe, spect, and the monks-problems, on which MLP outperforms all other methods.

5. Conclusion and topics of future research

We presented a novel method of decision tree induction based on formal concept analysis. In this method, the decision tree is constructed using a selection of nodes and edges from a modified line diagram of a concept lattice associated to input data. A heuristic based on a global information provided by the concept lattice, namely, the numbers of particularly defined lower formal concepts, is used to select splitting attributes. An experimental evaluation suggests good performance. Our method outperformed the instance-based method IB1 and is comparable to entropy-based methods ID3 and C4.5 and neural network method MLP.

Future research should focus on the following topics:

- The main novelty in our approach consists in using global information regarding natural clusters in input data that are represented by the concept lattice extracted from the data. Further research, both experimental and theoretical is necessary to better utilise this global information with respect to the design of good decision trees.
- Explore the possibility to compute a smaller number of formal concepts from which the nodes of a decision tree are constructed.

– Explore the problems of overfitting in data and incomplete data, i.e. data having missing values for some attributes in some records. These problems were not considered in this paper.
– Explore the possibility of incremental update of the induced decision trees via incremental methods of constructing concept lattices.
– Explore computational efficiency of our method. Namely, the use of the global information used for selecting the splitting attributes requires to compute a possibly large number of formal concepts.

Acknowledgements

Supported by grant No. 1ET101370417 of GA AV ČR, by institutional support, research plan MSM 6198959214, and by the Bilateral Scientific Cooperation Flanders–Czech Republic, Special Research Fund of Ghent University (Project No. 011S01106).

Notes

1. The paper is an extended version of a conference paper presented at CLA 2007, Montpellier, France, 24–26 October 2007.
2. Weka is a free software available at http://www.cs.waikato.ac.nz/ml/weka/

Notes on contributors

Radim Belohlavek is a Professor of Systems Science at Binghamton University-State University of New York. His academic interests are in the areas of uncertainty and information, fuzzy logic and fuzzy sets, data and knowledge engineering, data analysis, formal concept analysis, systems theory. Radim is a Senior Member of IEEE and a member of ACM and AMS. Before joining Binghamton University, he was a Professor and a Head of Department of Computer Science, Palacky University, Olomouc (Czech Republic).

Bernard De Baets leads KERMIT, the research unit *Knowledge-Based Systems*. He serves on the Editorial Boards of various international journals, in particular as co-editor-in-chief of Fuzzy Sets and Systems. Bernard coordinates EUROFUSE, the EURO Working Group on Fuzzy Sets, and is member of the Board of Directors of EUSFLAT, the Technical Committee on Artificial Intelligence and Expert Systems of IASTED, and of the Administrative Board of the Belgian OR Society.

Jan Outrata is an Assistant Professor at the Department of Computer Science, Palacky University in Olomouc, Czech Republic. He has obtained a PhD in Mathematics from Palacky University in 2006. His research interests include fuzzy logic and fuzzy sets, formal concept analysis and relational data analysis, clustering and knowledge engineering. He has authored over 20 papers in conference proceedings and journals including Journal of Computer and System Sciences, International Journal of General Systems, and International Journal of Foundations of Computer Science.

Vilem Vychodil is an Assistant Professor at SUNY Binghamton. He obtained a PhD in Mathematics in 2004 from Palacky University, Olomouc. His professional interests include fuzzy logic, fuzzy relational systems, relational data analysis, uncertainty in data, mathematical logic, and logical foundations of knowledge engineering. He has authored one monograph (Springer) and over 70 papers in conference proceedings and journals including Archives for Mathematical Logic, Mathematical Logic Quarterly, Logic Journal of IGPL, Journal of Experimental and Theoretical Artificial Intelligence, Fuzzy Sets and Systems, Journal of Multiple-Valued Logic and Soft Computing. Vilem Vychodil is a member of the ACM and IEEE.

References

Carpineto, C. and Romano, G., 1996. A lattice conceptual clustering system and its application to browsing retrieval. *Machine learning*, 24, 95–122.

Carpineto, C. and Romano, G., 2004. *Concept data analysis. Theory and applications*. New York: Wiley.

Dunham, M.H., 2003. *Data mining. Introductory and advanced topics*. Upper Saddle River, NJ: Prentice Hall.

Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P., 1996. From Data mining to knowledge discovery: an overview. *In*: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, eds. *Advances in knowledge discovery and data mining*. Melno Park, CA: AAAI Press, 3–33.

Fu, H., *et al.*, 2004. A comparative study of FCA-based supervised classification algorithms. *In*: P. Eklund, ed. *ICFCA 2004. Lecture notes in artificial intelligence 2961*, Berlin/Heidelberg: Springer-Verlag, 313–320.

Ganter, B. and Wille, R., 1999. *Formal concept analysis. Mathematical foundations*. Berlin: Springer.

Kuznetsov, S.O., 2004. Machine learning and formal concept analysis. *In*: P. Eklund, ed. *ICFCA 2004. Lecture notes in artificial intelligence 2961*, Berlin/Heidelberg: Springer-Verlag, 287–312.

Lindig, C., 2000. Fast concept analysis. *In*: G. Stumme, ed. *Working with conceptual structures – contributions to ICCS 2000*. Aachen: Shaker Verlag, 152–161.

Mephu Nguifo, E. and Njiwoua, P., 2001. IGLUE: a lattice-based constructive induction system. *Intelligent data analysis*, 5 (1), 73–91.

Mitchell, T.M., 1997. *Machine learning*. McGraw-Hill, New York, 1997.

Murthy, S.K., 1998. Automatic construction of decision trees from data. *Data mining and knowledge discovery*, 2, 345–389.

Newman, D.J., *et al.*, 1998. *UCI repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science. Available from: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Pasquier, N., *et al.*, 1999. Efficient mining of association rules using closed itemset lattices. *Information systems*, 24 (1), 25–46.

Quinlan, J.R., 1993. *C4.5: programs for machine learning*. San Francisco, CA: Morgan Kaufmann.

Quinlan, J.R., 1996. Learning decision tree classifiers. *ACM computing surveys*, 28 (1), 71–72.

Tan, P.N., Steinbach, M. and Kumar, V., 2006. *Introduction to data mining*. Boston, MA: Addison Wesley.

Witten, I.H. and Frank, E., 2005. *Data mining: practical machine learning tools and techniques*. 2nd ed. San Francisco, CA: Morgan Kaufmann.

Preprocessing input data for machine learning by FCA

Jan Outrata*

Department of Computer Science, Palacky University, Olomouc, Czech Republic
Tř. 17. listopadu 12, 771 46 Olomouc, Czech Republic
jan.outrata@upol.cz

Abstract. The paper presents an utilization of formal concept analysis in input data preprocessing for machine learning. Two preprocessing methods are presented. The first one consists in extending the set of attributes describing objects in input data table by new attributes and the second one consists in replacing the attributes by new attributes. In both methods the new attributes are defined by certain formal concepts computed from input data table. Selected formal concepts are so-called factor concepts obtained by boolean factor analysis, recently described by FCA. The ML method used to demonstrate the ideas is decision tree induction. The experimental evaluation and comparison of performance of decision trees induced from original and preprocessed input data is performed with standard decision tree induction algorithms ID3 and C4.5 on several benchmark datasets.

1 Introduction

Formal concept analysis (FCA) if ofted proposed to be used as a method for data preprocessing before the data is processed by another data mining or machine learning method [15, 8]. The results produced by these methods indeed depend on the structure of input data. In case of relational data described by objects and their attributes (object-attribute data) the structure of data is defined by the attributes and, more particularly, by dependencies between attributes. Data preprocessing in general then usually consits in transformation of the set of attributes to another set of attributes in order to enable the particular data mining or machine learning method to achieve better results [13, 14].

The paper presents a data preprocessing method utilizing formal concept analysis in a way that certain formal concepts are used to create new attributes describing the original objects. Selected formal concepts are so-called factor concepts obtained by boolean factor analysis, recently described by means of FCA in [1]. First, attributes defined by the concepts are added to the original set of attributes, extending the dimensionality of data. New attributes are supposed to aid the data mining or machine learning method. Second, the original attributes are replaced by the new attributes which usually means the reduction

* Supported by grant no. P202/10/P360 of the Czech Science Foundation

of dimensionality of data since the number of factor concepts is usually smaller than the number of original attributes. Here, a main question arises, whether the reduced number of new attributes can better describe the input objects for the subsequent data mining or machine learning method to produce better results.

There have been several attempts to transform the attribute space in order to improve the results of data mining and machine learning methods. From the variety of these methods we focus on decision tree induction. The most relevant to our paper is are methods known as constructive induction or feature construction [7], where new compound attributes are constructed from original attributes as conjunctions and/or disjunctions of the attributes [11] or arithmetic operations [12] or the new attributes are expressed in m-of-n form [9]. An oblique decision tree [10] is also connected to our approach in a sense that multiple attributes are used in the splitting condition (see section 3.1) instead of single attribute at a time. Typically linear combinations of attributes are looked for, e.g. [2]. Learning the condition is, however, computationally challenging.

Interestingly, we have not found any paper solely on this subject utilizing formal concept analysis. There have been several FCA-based approaches on construction of a whole learning model, commonly called lattice-based or concept-based machine learning approaches, e.g. [6], see [3] for a survey and comparison, but the usage of FCA to transform the attributes and create new attributes to aid another machine learning method is discussed very marginally or not at all. The present paper is thus a move to fill the gap.

The remainder of the paper is organized as follows. The next section contains preliminaries from FCA and introduction to boolean factor analysis, including the necessary tranformations between attribute and factor spaces. The main part of the paper is section 3 demonstrating the above sketched ideas on selected machine mearning method – decision tree induction. An experimental evaluation on selected data mining and machine learning benchmark datasets is provided in section 4. Finally, section 5 draws the conclusion.

2 Preliminaries

2.1 Formal Concept Analysis

In this section we summarize basic notions of FCA. For further information we refer to [4]. An object-attribute data table is identified with a triplet $\langle X, Y, I \rangle$ where X is a non-empty set of objects, Y is a non-empty set of attributes, and $I \subseteq X \times Y$ is an object-attribute relation. Objects and attributes correspond to table rows and columns, respectively, and $\langle x, y \rangle \in I$ indicates that object x has attribute y (table entry corresponding to row x and column y contains $\times$ or 1; otherwise it contains blank symbol or 0). In terms of FCA, $\langle X, Y, I \rangle$ is called a formal context. For every $A \subseteq X$ and $B \subseteq Y$ denote by $A^{\uparrow}$ a subset of Y and by $B^{\downarrow}$ a subset of X defined as

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

That is, $A^{\uparrow}$ is the set of all attributes from Y shared by all objects from A (and similarly for $B^{\downarrow}$). A formal concept in $\langle X, Y, I\rangle$ is a pair $\langle A, B\rangle$ of $A \subseteq X$ and $B \subseteq Y$ satisfying $A^{\uparrow} = B$ and $B^{\downarrow} = A$. That is, a formal concept consists of a set A (so-called extent) of objects which are covered by the concept and a set B (so-called intent) of attributes which are covered by the concept such that A is the set of all objects sharing all attributes from B and, conversely, B is the collection of all attributes from Y shared by all objects from A. Formal concepts represent clusters hidden in object-attribute data.

A set $\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \,|\, A^{\uparrow} = B, B^{\downarrow} = A\}$ of all formal concepts in $\langle X, Y, I\rangle$ can be equipped with a partial order $\leq$. The partial order models a subconcept-superconcept hierarchy, e.g. *dog* $\leq$ *mammal*, and is defined by

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \text{ iff } A_1 \subseteq A_2 \text{ (iff } B_2 \subseteq B_1).$$

$\mathcal{B}(X, Y, I)$ equipped with $\leq$ happens to be a complete lattice, called the concept lattice of $\langle X, Y, I\rangle$. The basic structure of concept lattices is described by the so-called basic theorem of concept lattices, see [4].

2.2 Boolean Factor Analysis

Boolean factor analysis is a matrix decomposition method which provides a representation of an object-attribute data matrix by a product of two different matrices, one describing objects by new attributes or factors, and the other describing factors by the original attributes [5]. Stated as the problem, the aim is to decompose an $n \times m$ binary matrix I into a boolean product $A \circ B$ of an $n \times k$ binary matrix A and a $k \times m$ binary matrix B with k as small as possible. Thus, instead of m original attributes, one aims to find k new attributes, called factors.

Recall that a binary (or boolean) matrix is a matrix whose entries are 0 or 1. A boolean matrix product $A \circ B$ of binary matrices A and B is defined by

$$(A \circ B)_{ij} = \bigvee_{l=1}^{k} A_{il} \cdot B_{lj},$$

where $\bigvee$ denotes maximum and $\cdot$ is the usual product. The interperetations of matrices A and B is: $A_{il} = 1$ means that factor l applies to object i and $B_{lj} = 1$ means that attribute j is one of the manifestations of factor l. Then $A \circ B$ says: "object i has attribute j if and only if there is a factor l such that l applies to i and j is one of the manifestations of l". As an example,

$$\begin{pmatrix} 1&1&0&0&0\\ 1&1&0&0&1\\ 1&1&1&1&0\\ 1&0&0&0&1 \end{pmatrix} = \begin{pmatrix} 1&0&0&1\\ 1&0&1&0\\ 1&1&0&0\\ 0&0&1&0 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&0&0\\ 0&0&1&1&0\\ 1&0&0&0&1\\ 0&1&0&0&0 \end{pmatrix}.$$

The (solution to the) problem of decomposition binary matrices was recently described by means of formal concept analysis [1]. The description lies in an observation that matrices A and B can be constructed from a set $\mathcal{F}$ of formal concepts of I. In particular, if $\mathcal{B}(X, Y, I)$ is the concept lattice associated to I,

with $X = \{1, \ldots, n\}$ and $Y = \{1, \ldots, m\}$, and

$$\mathcal{F} = \{\langle A_1, B_1 \rangle, \ldots, \langle A_k, B_k \rangle\} \subseteq \mathcal{B}(X, Y, I),$$

then for the $n \times k$ and $k \times m$ matrices $A_\mathcal{F}$ and $B_\mathcal{F}$ defined in such a way that the l-th column $(A_\mathcal{F})_{\cdot l}$ of $A_\mathcal{F}$ consists of the characteristic vector of A_l and the l-th row $(B_\mathcal{F})_{l\cdot}$ of $B_\mathcal{F}$ consists of the characteristic vector of B_l the following universality theorem holds:

Theorem 1. *For every I there is $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ such that $I = A_\mathcal{F} \circ B_\mathcal{F}$.*

Moreover, decompositions using formal concepts as factors are optimal in that they yield the least number of factors possible:

Theorem 2. *Let $I = A \circ B$ for $n \times k$ and $k \times m$ binary matrices A and B. Then there exists a set $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ of formal concepts of I with*

$$|\mathcal{F}| \leq k$$

such that for the $n \times |F|$ and $|F| \times m$ binary matrices $A_\mathcal{F}$ and $B_\mathcal{F}$ we have

$$I = A_\mathcal{F} \circ B_\mathcal{F}.$$

Formal concepts $\mathcal{F}$ in the above theorems are called factor concepts. Each factor concept determines a factor. For the constructive proof of the last theorem, examples and further results, we refer to [1].

2.3 Transformations between attribute and factor spaces

For every object i we can consider its representations in the m-dimensional Boolean space $\{0,1\}^m$ of original attributes and in the k-dimensional Boolean space $\{0,1\}^k$ of factors. In the space of attributes, the vector representing object i is the i-th row of the input data matrix I, and in the space of factors, the vector representing i is the i-th row of the matrix A.

Natural transformations between the space of attributes and the space of factors is described by the mappings $g : \{0,1\}^m \to \{0,1\}^k$ and $h : \{0,1\}^k \to \{0,1\}^m$ defined for $P \in \{0,1\}^m$ and $Q \in \{0,1\}^k$ by

$$(g(P))_l = \bigwedge_{j=1}^{m} (B_{lj} \to P_j), \tag{1}$$

$$(h(Q))_j = \bigvee_{l=1}^{k} (Q_l \cdot B_{lj}), \tag{2}$$

for $1 \leq l \leq k$ and $1 \leq j \leq m$. Here, $\to$ denotes the truth function of classical implication ($1 \to 0 = 0$, otherwise 1), $\cdot$ denotes the usual product, and $\bigwedge$ and $\bigvee$ denote minimum and maximum, respectively. (1) says that the l-th component of $g(P) \in \{0,1\}^k$ is 1 if and only if for every attribute j, $P_j = 1$ for all positions j for which $B_{lj} = 1$, i.e. the l-th row of B is included in P. (2) says that the j-th component of $h(Q) \in \{0,1\}^m$ is 1 if and only if there is factor l such that $Q_l = 1$ and $B_{lj} = 1$, i.e. attribute j is a manifestation of at least one factor from Q.

For results showing properties and describing the geometry behind the mappings g and h, see [1].

3 Boolean Factor Analysis and Decision Trees

The machine learning method which we use in this paper to demonstrate the ideas presented in section 1 is decision tree induction.

3.1 Decision Trees

Decision trees represent the most commonly used method in data mining and machine learning [13, 14]. A decision tree can be considered as a tree representation of a function over attributes which takes a finite number of values called class labels. The function is partially defined by a set of vectors (objects) of attribute values and the assigned class label, usually depicted by a table. An example function is depicted in Fig. 1. The goal is to construct a tree that approximates the function with a desired accuracy. This is called a decision tree induction. An induced decision tree is typically used for classification of objects into classes, based on the objects' attribute values. A good decision tree is supposed to classify well both objects described by the input data table as well as "unseen" objects.

Each non-leaf tree node of a decision tree is labeled by an attribute, called a splitting attribute for this node. Such a node represents a test, according to which objects covered by the node are split into v subcollections which correspond to v possible outcomes of the test. In the basic setting, the outcomes are represented by values of the splitting attribute. Leaf nodes of the tree represent collections of objects all of which, or the majority of which, have the same class label. An example of a decision tree is depicted in Fig. 4.

Many algorithms for the construction of decision trees were proposed in the literature, see e.g. [14]. A strategy commonly used consists of constructing a decision tree recursively in a top-down fashion, from the root node to the leaves, by successively splitting existing nodes into child nodes based on the splitting attribute. A critical point in this strategy is the selection of splitting attributes in nodes, for which many approaches were proposed. These include the well-known approaches based on entropy measures, Gini index, classification error, or other measures defined in terms of class distribution of the objects before and after splitting, see [14] for overviews.

Remark 1. In machine learning, and in decision trees at particular, the input data attributes are very often categorical attributes. To utilize FCA with the input data, we need to transform the categorical attributes to binary attributes because, in its basic setting, FCA works with binary attributes. A transformation of input data which consists in replacing non-binary attributes into binary ones is called conceptual scaling in FCA [4]. Note that we need not transform the class attribute, i.e. the attribute determining to which class the object belongs, because we transform the input attributes only in our data preprocessing method.

Throughout this paper, we use input data from Fig. 1 (top) to illustrate the data preprocessing. The data table contains sample animals described by

attributes *body temperature*, *gives birth*, *fourlegged*, *hibernates*, and *mammal*, with the last attribute being the class. After an obvious transformation (nominal scaling) of the input attributes, we obtain the data depicted in Fig. 1 (bottom). Boolean factor analysis which we use in our method is applied on data which we obtain after such transformation. For illustration, the decision tree induced from the data is depicted in Fig. 4 (left).

Name	*body temp.*	*gives birth*	*fourlegged*	*hibernates*	*mammal*
cat	warm	yes	yes	no	yes
bat	warm	yes	no	yes	yes
salamander	cold	no	yes	yes	no
eagle	warm	no	no	no	no
guppy	cold	yes	no	no	no

Name	*bt cold*	*bt warm*	*gb no*	*gb yes*	*fl no*	*fl yes*	*hb no*	*hb yes*	*mammal*
cat	0	1	0	1	0	1	1	0	yes
bat	0	1	0	1	1	0	0	1	yes
salamander	1	0	1	0	0	1	0	1	no
eagle	0	1	1	0	1	0	1	0	no
guppy	1	0	0	1	1	0	1	0	no

Fig. 1. Input data table (top) and corresponding data table for FCA (bottom)

3.2 Extending the collection of attributes

The first approach proposed in our data preprocessing method is the extension of the collection of attributes by new attributes which are created using boolean factor analysis. In praticular, the new attributes are represented by factors obtained from the decomposition of input data table.

Let $I \subseteq X \times Y$ be input data table describing objects $X = \{x_1, \ldots, x_n\}$ by binary attributes $Y = \{y_1, \ldots, y_m\}$. Considering I as a $n \times m$ binary matrix, we find a decomposition $I = A \circ B$ of I into the $n \times k$ matrix A describing objects by factors $F = \{f_1, \ldots, f_k\}$ and $k \times m$ matrix B explaining factors F by attributes. The decomposition of example data table in Fig. 1 is depicted in Fig. 2. The new collection of attributes Y' is then defined to be $Y' = Y \cup F$ and the extended data table $I' \subseteq X \times Y'$ is defined by $I' \cap (X \times Y) = I$ and $I' \cap (X \times F) = A$. Hence the new collection of attributes is the union of original attributes and factors and the extended data table is the apposition of original data table and the table representing the matrix describing objects by factors. Fig. 3 depicts the extended data table.

The key part is the decomposition of the original data table. In the decomposition of binary matrices the aim is to find the decomposition with the number of factors as small as possible. However, since the factors, as new attributes, are used in the process of decision tree induction in our application, we are looking

$$\begin{pmatrix} 0&1&0&1&0&1&1&0 \\ 0&1&0&1&1&0&0&1 \\ 1&0&1&0&0&1&0&1 \\ 0&1&1&0&1&0&1&0 \\ 1&0&0&1&1&0&1&0 \end{pmatrix} = \begin{pmatrix} 0&0&1&0&0&1 \\ 0&0&1&0&1&0 \\ 0&0&0&1&0&0 \\ 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \end{pmatrix} \circ \begin{pmatrix} 0&1&1&0&1&0&1&0 \\ 1&0&0&1&1&0&1&0 \\ 0&1&0&1&0&0&0&0 \\ 1&0&1&0&0&1&0&1 \\ 0&1&0&1&1&0&0&1 \\ 0&1&0&1&0&1&1&0 \end{pmatrix}$$

Fig. 2. Boolean matrix decomposition of input data in Fig. 1

Name	*bc*	*bw*	*gn*	*gy*	*fn*	*fy*	*hn*	*hy*	f_1	f_2	f_3	f_4	f_5	f_6	*mammal*
cat	0	1	0	1	0	1	1	0	0	0	1	0	0	1	yes
bat	0	1	0	1	1	0	0	1	0	0	1	0	1	0	yes
salamander	1	0	1	0	0	1	0	1	0	0	0	1	0	0	no
eagle	0	1	1	0	1	0	1	0	1	0	0	0	0	0	no
guppy	1	0	0	1	1	0	1	0	0	1	0	0	0	0	no

Fig. 3. Extended data table for input data in Fig. 1

also for the factors which have a good "decision ability", i.e. that the factors are good candidates to be splitting attributes. To compute the decomposition we can use the algorithms presented in [1], with modified criterion of optimality of computed factors. In short, the algorithms apply a greedy heuristic approach to search in the space of all formal concepts for the factor concepts which cover the largest area of still uncovered 1s in the input data table. The criterion function of optimality of a factor is thus the "cover ability" of the corresponding factor concept, in particular the number of uncovered 1s in the input data table which are covered by the concept, see [1]. The function value is, for the purposes of this paper, translated to the interval $[0, 1]$ (with the value of 1 meaning the most optimal) by dividing the value by the total number of still uncovered 1s in the data table.

The new criterion function $c\colon 2^{X\times Y} \to [0, 1]$ of optimality of factor concept $\langle A, B\rangle$ is:

$$c(\langle A, B\rangle) = w \cdot c_A(\langle A, B\rangle) + (1 - w) \cdot c_B(\langle A, B\rangle), \tag{3}$$

where $c_A(\langle A, B\rangle) \in [0, 1]$ is the original criterion function of the "cover ability" of factor concept $\langle A, B\rangle$, $c_B(\langle A, B\rangle) \in [0, 1]$ is a criterion function of the "decision ability" of factor concept $\langle A, B\rangle$ and w is a weight of preference among the functions c_A and c_B. Let us focus on the function c_B. The function measures the goodness of the factor, defined by the factor concept, as splitting attribute. As was mentioned in section 3.1, in decision trees, a common approaches to selection of splitting attribute are based on entropy measures. In these approaches, an attribute is the better splitting attribute the lower is the weighted sum of entropies of subcollections of objects after splitting the objects based on the

attribute. We thus design the function c_B to be such a measure:

$$c_B(\langle A, B\rangle) = 1 - \left(\frac{|A|}{|X|} \cdot \frac{E(\text{class}|A)}{-\log_2 \frac{1}{|V(\text{class}|A)|}} + \frac{|X \setminus A|}{|X|} \cdot \frac{E(\text{class}|X \setminus A)}{-\log_2 \frac{1}{|V(\text{class}|X \setminus A)|}} \right), \tag{4}$$

where $V(\text{class}|A)$ is the set of class labels assigned to objects A and $E(\text{class}|A)$ is the entropy of objects A based on the class defined as usual by:

$$E(\text{class}|A) = -\sum_{l \in V(\text{class}|A)} p(l|A) \cdot \log_2 p(l|A), \tag{5}$$

where $p(l|A)$ is the fraction of objects A with assigned class label l. The value of $-\log_2 \frac{1}{|V(\text{class}|A)|}$ in (4) is the maximal possible value of entropy of objects A in the case the class labels $V(\text{class}|A)$ are assigned to objects A evenly and the purpose of it is to normalize the value of c_B to the interval $[0, 1]$. Note that we put $\frac{0}{0} = 0$ in calculations in (4).

Now, having the extended data table $I' \subseteq X \times (Y \cup F)$ containing new attributes F, the decision tree is induced from the extended data table instead of the original data table I. The class labels assigned to objects remain unchanged, see Fig. 3. For ilustration, the decision tree induced from data table in Fig. 3 is depicted in Fig. 4 (right). We can see that the data can be decided by a single attribute, namely, factor f_3 the manifestations of which are original attributes *bt warm* and *gb yes*. Factor f_3, as the combination of the two attributes, is a better splitting attribute in decision tree induction than the two attributes alone.

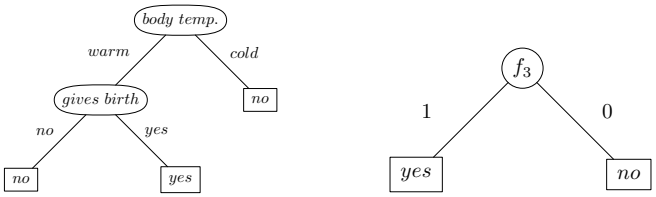

Fig. 4. The decision trees induced from original data table in Tab. 1 (left) and from extended data table in Tab. 3 (right)

The resulted decision tree is used as follows. When classifying an object x described by original attributes Y as a vector $P_x \in \{0, 1\}^m$ in the (original) attribute space, we first need to compute the description of the object by new attributes/factors F as a vector $g(P_x) \in \{0, 1\}^k$ in the factor space. This is accomplished by (1) using the matrix B explaining factors in terms of original attributes. The object described by concatenation of P_x and $g(P_x)$ is then classified by the decision tree in a usual way.

For instance, an object described by original attributes Y as vector $(10011010)_Y$ is described by factors F as vector $(010000)_F$. The object described by concate-

nation of these two vectors is classified by class label *no* by the decision tree in Fig. 4 (right).

3.3 Reducing the collection of attributes

The second approach consits in the replacement of original attributes by factors, i.e. discarding the original data table. Hence the new collection of attributes Y' is defined to be $Y' = F$ and the new data table $I' \subseteq X \times Y'$ is put to $I' = A$, where A is the $n \times k$ binary matrix describing objects by factor resulting from the decomposition $I = A \circ B$ of input data table I. Hence the new reduced data table for example data in Fig. 1 is a table depicted in Fig. 3 restricted to attributes $f_1, \ldots, f_6$.

Since the number of factors is usually smaller than the number of attributes, see [1], this transformation usually leads to the reduction of dimensionality of data. However, the transformation of objects from attribute space to the factor space is not an injective mapping. In particular, the mapping g from attribute vectors to factor vectors maps large convex sets of objects to the same points in the factor space, see [1] for details. Namely, for two distinct objects $x_1, x_2 \in X$ with different attributes, i.e. described by different vectors in the space of attributes, $P_{x_1} \neq P_{x_2}$, which have different class labels assigned, $\mathrm{class}(x_1) \neq \mathrm{class}(x_2)$, the representation of both x_1, x_2 by vectors in the factor space is the same, $g(P_{x_1}) = g(P_{x_2})$.

Consider the relation $\ker(g)$ (the kernel relation of g) describing such a situation. The class $[x]_{\ker(g)} \in X/\ker(g)$ for an object $x \in X$ contains objects represented in (original) attribute space which are mapped to the same object x represented in factor space. The class label assigned to each object $x \in X$ in the new data table I' is the majority class label for the class $[x]_{\ker(g)} \in X/\ker(g)$ defined as follows: a class label l is a majority class label for $[x]_{\ker(g)}$ if l is assigned to the most of objects from $[x]_{\ker(g)}$, i.e. if $l = \mathrm{class}(x_1)$ for $x_1 \in [x]_{\ker(g)}$ such that for each $x' \in [x]_{\ker(g)}$ it holds:

$$|\{x_2 \in [x]_{\ker(g)} \,|\, \mathrm{class}(x_2) = l\}| \geq |\{x_2 \in [x]_{\ker(g)} \,|\, \mathrm{class}(x_2) = \mathrm{class}(x')\}|.$$

Finally, the decision tree is induced from the transformed data table $I' \subseteq X \times F$, where class labels assigned to each object $x \in X$ is the majority class label for the class $[x]_{\ker(g)} \in X/\ker(g)$. Similarily as in the first approach in section 3.2, when classifying an object x described by original attributes Y as a vector $P_x \in \{0,1\}^m$ in the (original) attribute space, we first compute the description of the object by factors F as a vector $g(P_x) \in \{0,1\}^k$ in the factor space. The object described by $g(P_x)$ is classified by the decision tree. In our example, the decision tree induced from reduced data table (the table in Fig. 3 restricted to attributes $f_1, \ldots, f_6$) is the same as the tree induced from the extended data table, i.e. the tree depicted in Fig. 4 (right).

4 Experimental Evaluation

We performed series of experiments to evaluate our data preprocessing method. The experiments consist in comparing the performance of created machine learning models (e.g. decision trees) induced from original and preprocessed input data. In the comparison we used reference decision tree algorithms ID3 and C4.5 [13] (entropy and information gain based) and also an instance based learning method (IB1). The algorithms were borrowed and run from Weka [1], a software package that contains implementations of machine learning and data mining algorithms in Java. Default Weka's parameters were used for the algorithms.

Table 1. Characteristics of datasets used in experiments

Dataset	*No. of attributes (binary)*	*No. of objects*	*Class distribution*
breast-cancer	9(51)	277	196/81
kr-vs-kp	36(74)	3196	1669/1527
mushroom	21(125)	5644	3488/2156
tic-tac-toe	9(27)	958	626/332
vote	16(32)	232	124/108
zoo	15(30)	101	41/20/5/13/4/8/10

The experiments were done on selected public real-world datasets from UCI Machine Learning Repository. The selected datasets are from different areas (medicine, biology, zoology, politics, games). All the datasets contain only categorical attributes with one class label attribute and the datasets were cleared of objects containing missing values. Basic characteristics of the datasets are depicted in Tab. 1. The numbers of attributes are of original categorical attributes and, in brackets, of binary attributes after nominal scaling (see remark 1). The experiments were done using the 10-fold stratified cross-validation test. The following results are of averaging 10 execution runs on each dataset with randomly ordered records.

Due to the limited scope of the paper we show only the results of data preprocessing by reducing the original attributes to factors and the results for adding the factors to the collection of attributes are postponed to the full version of the paper. The results are depicted in Tab. 2. The tables show ratios of the average percentage rates of correct classifications for preprocessed data and original data, i.e. the values indicate the increase factor of correct classifications for preprocessed data. The values are for both training (upper number in the table cell) and testing (lower number) datasets for each algorithm and dataset being compared, plus the average over all datasets. In the case of top table the

[1] Waikato Environment for Knowledge Analysis, available at `http://www.cs.waikato.ac.nz/ml/weka/`

criterion of optimality of generated factors (3) was set to the original criterion function of the "cover ability" of factor concept, i.e. the original criterion used in the algorithms from [1]. This corresponds to setting $w = 1$ in (3). In the case of bottom table the criterion of optimality of generated factors was changed to the function of the "decision ability" described in section 3.2, i.e. $w = 0$ in (3).

Table 2. Classification accuracy for datasets from Tab. 1, for $w = 1$ (top) and $w = 0$ (bottom table) in (3)

training % *testing %*	*breast-cancer*	*kr-vs-kp*	*mushroom*	*tic-tac-toe*	*vote*	*zoo*	*average*
ID3	1.020	1.000	1.000	1.000	1.000	1.018	1.006
	1.159	0.993	1.000	1.123	0.993	0.962	1.038
C4.5	1.031	0.998	1.000	1.028	0.998	1.006	1.010
	0.989	0.994	1.000	1.092	0.994	0.940	1.002
IB1	1.020	1.000	1.000	1.000	1.000	1.020	1.007
	0.970	1.017	1.000	1.000	1.005	0.965	0.993

training % *testing %*	*breast-cancer*	*kr-vs-kp*	*mushroom*	*tic-tac-toe*	*vote*	*zoo*	*average*
ID3	1.020	1.000	1.000	1.000	1.000	1.018	1.006
	1.153	1.000	1.000	1.157	1.017	0.980	1.051
C4.5	1.047	1.000	1.000	1.033	1.000	1.006	1.014
	1.035	0.998	1.000	1.138	1.007	0.958	1.023
IB1	1.020	1.000	1.000	1.000	1.000	1.020	1.007
	0.951	1.083	1.000	1.213	1.033	0.967	1.041

We can see that while not inducing worse learning model at average on training datasets the methods have better performance at average on testing dataset for input data preprocessed by our methods (with the exception of dataset zoo which has more than two values of class attribute). For instance, ID3 method has better performance by 3.8 % (5.4 % without zoo) for criterion of optimality of generated factors being the original criterion function of the "cover ability" of factor concept, while for criterion of optimality of generated factors being the function of the "decision ability" the performance is better by 5.1 % (6.5 % without zoo). The results for adding the factors to the collection of attributes are very similar, with ±1 % difference to the results for reducing the original attributes to factors, with the exception of dataset zoo, where the difference was +4 %.

5 Conclusion

We presented two methods of preprocessing input data to machine learning based on formal concept analysis (FCA). In the first method, the collection of attributes describing objects is extended by new attributes while in the second method, the original attributes are replaced by the new attributes. Both methods utilize boolean factor analysis, recently described by FCA, in that the new attributes are defined as factors computed from input data. The number of factors is usually smaller than the number of original attributes. The methods were demonstrated on the induction of decision trees and an experimental evaluation indicates usefullness of such preprocessing of data: the decision trees induced from preprocessed data outperformed decision trees induced from original data for two entropy-based methods ID3 and C4.5.

References

1. Belohlavek R., Vychodil V.: Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. System Sci* **76**(1)(2010), 3-20.
2. Breiman L., Friedman J. H., Olshen R., Stone C. J.: *Classification and Regression Trees.* Chapman & Hall, NY, 1984.
3. Fu H., Fu H., Njiwoua P., Mephu Nguifo E.: A comparative study of FCA-based supervised classification algorithms. In: Proc. ICFCA 2004, *LNAI* **2961**, 2004, pp. 313–320.
4. Ganter B., Wille R.: *Formal Concept Analysis. Mathematical Foundations.* Springer, Berlin, 1999.
5. Kim K. H.: *Boolean Matrix Theory and Applications.* M. Dekker, 1982.
6. Kuznetsov S. O.: Machine learning and formal concept analysis. In: Proc. ICFCA 2004, *LNAI* **2961**, 2004, pp. 287–312.
7. Michalski R. S.: A theory and methodology of inductive learning. *Artificial Intelligence* **20**(1983), 111–116.
8. Missaoui R., Kwuida L.: What Can Formal Concept Analysis Do for Data Warehouses? In Proc. ICFCA 2009, *LNAI* **5548**, 2009, 58–65.
9. Murphy P. M., Pazzani M. J.: ID2-of-3: constructive induction of M-of-N concepts for discriminators in decision trees. In Proc. of the Eight Int. Workshop on Machine Learning, 1991, 183–187.
10. Murthy S. K., Kasif S., Salzberg S.: A system for induction of oblique decision trees. *J. of Artificial Intelligence Research* **2**(1994), 1–33.
11. Pagallo G., Haussler D.: Boolean feature discovery in empirical learning. *Machine Learning* **5**(1)(1990), 71–100.
12. Piramuthu S., Sikora R. T.: Iterative feature construction for improving inductive learning algorithms. *Expert Systems with Applications* **36**(2, part 2)(2009), 3401–3406.
13. Quinlan J. R.: *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
14. Tan P.-N., Steinbach M., Kumar V.: *Introduction to Data Mining.* Addison Wesley, Boston, MA, 2006.
15. Valtchev P., Missaoui R., Godin R.: Formal concept analysis for knowledge discovery and data mining: The new challenges. In: Proc. ICFCA 2004, *LNAI* **2961**, 2004, pp. 352–371.

Boolean factor analysis for data preprocessing in machine learning

Jan Outrata
Department of Computer Science
Palacky University
Olomouc, Czech Republic
Email: jan.outrata@upol.cz

***Abstract*—We present two input data preprocessing methods for machine learning (ML). The first one consists in extending the set of attributes describing objects in input data table by new attributes and the second one consists in replacing the attributes by new attributes. The methods utilize formal concept analysis (FCA) and boolean factor analysis, recently described by FCA, in that the new attributes are defined by so-called factor concepts computed from input data table. The methods are demonstrated on decision tree induction. The experimental evaluation and comparison of performance of decision trees induced from original and preprocessed input data is performed with standard decision tree induction algorithms ID3 and C4.5 on several benchmark datasets.**

***Keywords*-data preprocessing; machine learning; decision trees; matrix decomposition; formal concept**

I. Introduction

Input data is often subject to some sort of data preprocessing before the data is processed by a data mining or machine learning method. In common case of relational data described by objects and their attributes (object-attribute data) data preprocessing usually consists in transformation of the set of attributes to another set of attributes in order to enable the particular data mining or machine learning method to achieve better results [13], [14].

The paper presents a data preprocessing method utilizing boolean factor analysis (BFA), recently described by means of formal concept analysis (FCA) [1]. The utilization consists in creating new attributes describing the original objects. The new attributes are defined in terms of so-called factor concepts obtained by BFA using FCA. We show two methods. First, new attributes are added to the original set of attributes, extending the dimensionality of data. New attributes are supposed to aid the data mining or machine learning method. Second, the original attributes are replaced by the new attributes which usually means the reduction of dimensionality of data. Here, a main question arises, whether the reduced number of new attributes can better describe the input objects for the subsequent data mining or machine learning method to produce better results.

There have been several attempts to transform the attribute space in order to improve the results of data mining and machine learning methods. We focus on decision tree induction. The most relevant to our paper is a method known as constructive induction [7], where new compound attributes are constructed from original attributes as conjunctions and/or disjunctions of the attributes [12] or the new attributes are expressed in m-of-n form [8]. An oblique decision tree [10] is also connected to our approach in a sense that multiple attributes are used in the splitting condition (see section III-A) instead of single attribute at a time, see e.g. [2]. Learning the condition is, however, computationally challenging.

II. Preliminaries

A. Formal Concept Analysis

Formal concept analysis (FCA) is a method for analysis of object-attribute data [3], [4]. Such data is usually described by a table with rows and columns representing objects and attributes, respectively, and with table entries containing attribute values which the objects have. See Fig. 1 (bottom) for an example of such a table. In the basic setting, FCA deals with binary attributes, i.e. every attribute applies or does not apply to a particular object. Many-valued attributes, such as nominal and ordinal attributes, are transformed to binary ones using so-called conceptual scaling. One of the outputs of FCA is a concept lattice, a partially ordered collection of particular clusters called formal concepts. Due to lack of space we refer for formal description and further information on FCA to [3], [4].

B. Boolean Factor Analysis

The aim of boolean factor analysis (BFA) is to decompose an $n \times m$ binary matrix I into a boolean product $A \circ B$ of an $n \times k$ binary matrix A and a $k \times m$ binary matrix B with k as small as possible. Thus, instead of m original attributes, one aims to find k new attributes, called factors.

Recall that a binary (or boolean) matrix is a matrix whose entries are 0 or 1. A boolean matrix product $A \circ B$ of binary matrices A and B is defined by

$$(A \circ B)_{ij} = \bigvee_{l=1}^{k} A_{il} \cdot B_{lj},$$

where $\bigvee$ denotes maximum and $\cdot$ is the usual product. The interpretation of matrices A and B is: $A_{il} = 1$ means that factor l applies to object i and $B_{lj} = 1$ means that attribute j is one of the manifestations of factor l. Then $A \circ B$ says: "object i has attribute j if and only if there is a factor l such that l applies to i and j is one of the manifestations of l".

The (solution to the) problem of decomposing binary matrices was recently described by means of formal concept analysis [1]. The description lies in an observation that matrices A and B can be constructed from a set $\mathcal{F}$ of formal concepts of I. In particular, if $\mathcal{B}(X, Y, I)$ is the concept lattice of I, with $X = \{1, \ldots, n\}$ and $Y = \{1, \ldots, m\}$, and

$$\mathcal{F} = \{\langle A_1, B_1\rangle, \ldots, \langle A_k, B_k\rangle\} \subseteq \mathcal{B}(X, Y, I),$$

then for the $n \times k$ and $k \times m$ matrices $A_{\mathcal{F}}$ and $B_{\mathcal{F}}$ defined in such a way that the l-th column $(A_{\mathcal{F}})_{\_l}$ of $A_{\mathcal{F}}$ consists of the characteristic vector of A_l and the l-th row $(B_{\mathcal{F}})_{l\_}$ of $B_{\mathcal{F}}$ consists of the characteristic vector of B_l, the following universality theorem holds:

Theorem 1: For every I there is $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ such that $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Moreover, decompositions using formal concepts as factors are optimal in that they yield the least number of factors possible:

Theorem 2: Let $I = A \circ B$ for $n \times k$ and $k \times m$ binary matrices A and B. Then there exists a set $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ of formal concepts of I with $|\mathcal{F}| \leq k$ such that for the $n \times |F|$ and $|F| \times m$ binary matrices $A_{\mathcal{F}}$ and $B_{\mathcal{F}}$ we have $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Formal concepts $\mathcal{F}$ are called factor concepts. Each factor concept determines a factor. For the constructive proof of the last theorem, examples and further results, we refer to [1].

C. Transformations between attribute and factor spaces

For every object i we can consider its representations in the m-dimensional Boolean space $\{0, 1\}^m$ of original attributes and in the k-dimensional Boolean space $\{0, 1\}^k$ of factors.

Natural transformations between the space of attributes and the space of factors is described by the mappings $g : \{0, 1\}^m \to \{0, 1\}^k$ and $h \colon \{0, 1\}^k \to \{0, 1\}^m$ defined by

$$(g(P))_l = \bigwedge_{j=1}^{m} (B_{lj} \to P_j), \quad (h(Q))_j = \bigvee_{l=1}^{k} (Q_l \cdot B_{lj}), \quad (1)$$

for $1 \leq l \leq k$ and $1 \leq j \leq m$. Here, $\to$ denotes the truth function of classical implication ($1 \to 0 = 0$, otherwise 1), $\cdot$ denotes the usual product, and $\bigwedge$ and $\bigvee$ denote minimum and maximum, respectively. (1, left) says that the l-th component of $g(P) \in \{0, 1\}^k$ is 1 if and only if the l-th row of B is included in P. (1, right) says that the j-th component of $h(Q) \in \{0, 1\}^m$ is 1 if and only if attribute j is a manifestation of at least one factor from Q.

For results showing properties and describing the geometry behind the mappings g and h, see [1].

III. Boolean Factor Analysis and Decision Trees

The machine learning method which we use to demonstrate the ideas presented in section I is decision tree induction.

A. Decision Trees

Decision trees represent the most commonly used method in data mining and machine learning [13], [14]. A decision tree can be considered as a tree representation of a function over attributes which takes a finite number of values called class labels. The function is partially defined by a set of vectors (objects) of attribute values and the assigned class label, usually depicted by a table. An example function is depicted in Fig. 1. The goal is to construct a tree that approximates the function with a desired accuracy. An induced decision tree is typically used for classification of objects into classes. A good decision tree is supposed to classify well both objects described by the input data table as well as "unseen" objects.

Name	*body temp.*	*gives birth*	*fourlegged*	*hibernates*	*mammal*
cat	warm	yes	yes	no	yes
bat	warm	yes	no	yes	yes
salamander	cold	no	yes	yes	no
eagle	warm	no	no	no	no
guppy	cold	yes	no	no	no

Name	*bt cold*	*bt warm*	*gb no*	*gb yes*	*fl no*	*fl yes*	*hb no*	*hb yes*	*mammal*
cat	0	1	0	1	0	1	1	0	yes
bat	0	1	0	1	1	0	0	1	yes
salamander	1	0	1	0	0	1	0	1	no
eagle	0	1	1	0	1	0	1	0	no
guppy	1	0	0	1	1	0	1	0	no

Fig. 1. Input data table (top) and corresponding data table for FCA (bottom)

Each non-leaf node of a decision tree is labeled by an attribute, called a splitting attribute for this node. Such a node represents a test, according to which objects covered by the node are split into v sub-collections which correspond to v possible outcomes of the test. In the basic setting, the outcomes are represented by values of the splitting attribute. Leaf nodes of the tree represent collections of objects all of which, or the majority of which, have the same class label. An example of a decision tree is depicted in Fig. 4.

Many algorithms for the construction of decision trees were proposed in the literature, see e.g. [14]. A strategy commonly used consists of constructing a tree recursively from the root node to the leaves, by successively splitting existing nodes into child nodes based on the splitting attribute. The many well-known approaches proposed for the selection of splitting attributes are based on entropy measures, Gini index, classification error, or other measures defined in terms of class distribution of the objects before and after splitting, see [9], [14] for overviews.

Remark 1: In machine learning the input data attributes are very often categorical attributes. To utilize FCA with the input data, we need to transform the categorical attributes to binary attributes using conceptual scaling [4]. Note that we need not transform the class attribute because we manipulate the input attributes only in our data preprocessing methods.

Throughout this paper, we use input data from Fig. 1 (top) to illustrate the data preprocessing. The data table contains sample animals described by five attributes, with the last attribute being the class. After an obvious transformation (nominal scaling) we obtain the data depicted in Fig. 1 (bottom). Boolean factor analysis is applied on the transformed data. For illustration, the decision tree induced from the data is depicted in Fig. 4 (left).

$$\begin{pmatrix} 0&1&0&1&0&1&1&0\\ 0&1&0&1&1&0&0&1\\ 1&0&1&0&0&1&0&1\\ 0&1&1&0&1&0&1&0\\ 1&0&0&1&1&0&1&0 \end{pmatrix} = \begin{pmatrix} 0&0&1&0&0&1\\ 0&0&1&0&1&0\\ 0&0&0&1&0&0\\ 1&0&0&0&0&0\\ 0&1&0&0&0&0 \end{pmatrix} \circ \begin{pmatrix} 0&1&1&0&1&0&1&0\\ 1&0&0&1&1&0&1&0\\ 0&1&0&1&0&0&0&0\\ 1&0&1&0&0&1&0&1\\ 0&1&0&1&1&0&0&1\\ 0&1&0&1&0&1&1&0 \end{pmatrix}$$

Fig. 2. Boolean matrix decomposition of input data in Fig. 1

Name	*bc*	*bw*	*gn*	*gy*	*fn*	*fy*	*hn*	*hy*	f_1	f_2	f_3	f_4	f_5	f_6	*mammal*
cat	0	1	0	1	0	1	1	0	0	0	1	0	0	1	yes
bat	0	1	0	1	1	0	0	1	0	0	1	0	1	0	yes
salamander	1	0	1	0	0	1	0	1	0	0	0	1	0	0	no
eagle	0	1	1	0	1	0	1	0	1	0	0	0	0	0	no
guppy	1	0	0	1	1	0	1	0	0	1	0	0	0	0	no

Fig. 3. Extended data table for input data in Fig. 1

B. Extending the collection of attributes

Our first proposed data preprocessing method is the extension of the collection of attributes by new attributes which are created using BFA.

Let $I \subseteq X \times Y$ be input data table describing objects $X = \{x_1, \ldots, x_n\}$ by binary attributes $Y = \{y_1, \ldots, y_m\}$. Considering I as a $n \times m$ binary matrix, we find a decomposition $I = A \circ B$ of I into the $n \times k$ matrix A describing objects by factors $F = \{f_1, \ldots, f_k\}$ and $k \times m$ matrix B explaining factors F by attributes. The decomposition of example data table in Fig. 1 is depicted in Fig. 2. The new collection of attributes Y' is then defined to be $Y' = Y \cup F$ and the extended data table $I' \subseteq X \times Y'$ is defined by $I' \cap (X \times Y) = I$ and $I' \cap (X \times F) = A$. Fig. 3 depicts the extended data table.

Recall that in the decomposition of binary matrices the aim is to find the decomposition with the number of factors as small as possible. To compute the decomposition one can use the algorithms presented in [1]. In short, the algorithms apply a greedy heuristic approach to search in the space of all formal concepts for the factor concepts which cover the largest area of still uncovered 1s in the input data table. The criterion function of optimality of a factor is thus the "cover ability" of the corresponding factor concept, in particular the number of uncovered 1s which are covered by the concept, see [1]. The function value is translated to the interval $[0, 1]$ (with the value of 1 meaning the most optimal). However, since the factors, as new attributes, are used in the process of decision tree induction in our application, we are looking also for the factors which have a good "decision ability", i.e. that the factors are good candidates to be splitting attributes.

The new criterion function $c\colon 2^{X \times Y} \to [0, 1]$ of optimality of factor concept $\langle A, B \rangle$ is:

$$c(\langle A, B \rangle) = w \cdot c_A(\langle A, B \rangle) + (1 - w) \cdot c_B(\langle A, B \rangle), \quad (2)$$

where $c_A(\langle A, B \rangle) \in [0, 1]$ is the original criterion function of the "cover ability", $c_B(\langle A, B \rangle) \in [0, 1]$ is a criterion function of the "decision ability" and w is a weight of preference among the functions c_A and c_B. Let us focus on the function c_B, which measures the goodness of the factor, defined by the factor concept, as splitting attribute. In common approaches to selection of splitting attribute based on entropy measures, an attribute is the better splitting attribute the lower is the weighted sum of entropies of sub-collections of objects after splitting the objects based on the attribute. We thus design the function c_B to be such a measure:

$$c_B(\langle A, B \rangle) = 1 - \left(\frac{|A|}{|X|} \cdot \frac{E(\text{class}|A)}{-\log_2 \frac{1}{|V(\text{class}|A)|}} + \frac{|X \setminus A|}{|X|} \cdot \frac{E(\text{class}|X \setminus A)}{-\log_2 \frac{1}{|V(\text{class}|X \setminus A)|}} \right), \quad (3)$$

where $V(\text{class}|A)$ is the set of class labels assigned to objects A and $E(\text{class}|A)$ is the entropy of objects A based on the class defined as usual. Note that we put $\frac{0}{0} = 0$ in calculations in (3).

Now, the decision tree is induced from the extended data table $I' \subseteq X \times (Y \cup F)$ instead of the original data table I. The class labels assigned to objects remain unchanged, see Fig. 3. For illustration, the decision tree induced from data table in Fig. 3 is depicted in Fig. 4 (right). We can see that the data can be decided by a single attribute, namely, factor f_3 the manifestations of which are original attributes *bt warm* and *gb yes*.

When classifying an object x described by original attributes Y as a vector $P_x \in \{0, 1\}^m$ in the (original) attribute space, we first compute the description of the object by new attributes/factors F as a vector $g(P_x) \in \{0, 1\}^k$ in the factor space by (1, left). The object described by concatenation of P_x and $g(P_x)$ is then classified by the decision tree in a usual way.

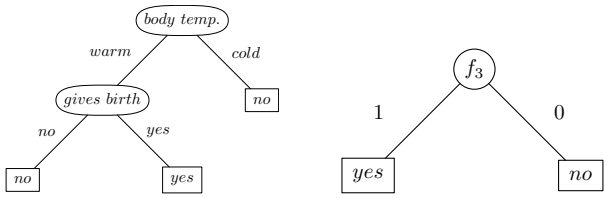

Fig. 4. The decision trees induced from original data table in Tab. 1 (left) and from extended data table in Tab. 3 (right)

C. Reducing the collection of attributes

The second of our two methods consists in the replacement of original attributes by factors. Hence the new collection of attributes Y' is defined to be $Y' = F$ and the new data table $I' \subseteq X \times Y'$ is put to $I' = A$. The new reduced data table for example data in Fig. 1 is thus a table depicted in Fig. 3 restricted to attributes $f_1, \ldots, f_6$.

Since the number of factors is usually smaller than the number of attributes, see [1], the transformation of objects from attribute space to the factor space usually leads to the reduction of dimensionality of data. However, the transformation is not an injective mapping, see [1] for details. Namely, for two distinct objects $x_1, x_2 \in X$ described by different vectors in the space of attributes and different class labels assigned, the representation of both x_1, x_2 by vectors in the factor space might be the same.

Consider the relation $\ker(g)$ (the kernel relation of g) describing such a situation. We select the class label assigned

TABLE I
CHARACTERISTICS OF DATASETS USED IN EXPERIMENTS

Dataset	*No. of attributes (binary)*	*No. of objects*	*Class distribution*
breast-cancer	9 (51)	277	196/81
kr-vs-kp	36 (74)	3196	1669/1527
mushroom	21 (125)	5644	3488/2156
tic-tac-toe	9 (27)	958	626/332
vote	16 (32)	232	124/108

to each object $x \in X$ in the new data table I' to be the majority class label for the class $[x]_{\ker(g)} \in X/\ker(g)$.

Finally, the decision tree is induced from the transformed data table $I' \subseteq X \times F$. Similarly as in the first method from section III-B, when classifying an object x, we first compute the description of the object by factors F as a vector $g(P_x) \in \{0,1\}^k$ in the factor space and the object described by $g(P_x)$ is classified by the decision tree. In our example, the decision tree induced from reduced data table (the table in Fig. 3 restricted to attributes $f_1, \ldots, f_6$) is the same as the tree induced from the extended data table, i.e. the tree depicted in Fig. 4 (right).

IV. EXPERIMENTAL EVALUATION

The experiments consist in comparing the performance of created machine learning models (e.g. decision trees) induced from original and preprocessed input data. In the comparison we used reference decision tree algorithms ID3 and C4.5 [13] (entropy and information gain based) and also an instance based learning method (IB1). The algorithms were borrowed and run from Weka [15].

Selected public real-world datasets from UCI Machine Learning Repository [11] we used in the experiments. Basic characteristics of the datasets are depicted in Tab. I. The numbers of attributes are of original categorical attributes and, in brackets, of binary attributes after nominal scaling (see remark 1). The experiments were done using the 10-fold stratified cross-validation test. The following results are of averaging 10 execution runs of model learning on each dataset fold with randomly ordered records.

Due to the very limited scope of the paper we show only the results of data preprocessing by reducing the original attributes to factors. The results are depicted in Tab. II. The tables show ratios of the average percentage rates of correct classifications for preprocessed data and original data, i.e. the values indicate the increase factor of correct classifications for preprocessed data. In the case of table on the left the criterion of optimality of generated factors (2) was set to the original criterion function of the "cover ability" of factor concept, which corresponds to setting $w = 1$ in (2). In the case of table on the right the criterion was changed to the function of the "decision ability" described in section III-B, i.e. $w = 0$.

We can see that, for input data preprocessed by our methods, while not inducing worse learning model at average on training datasets the induction methods have better performance at average on testing datasets. For instance, ID3 method has better performance by $5.4\,\%$ for criterion of optimality of generated factors being the original criterion function of the "cover ability" of factor concept, while for the criterion being the function of the "decision ability" the performance is better by $6.5\,\%$. We just note that the results for adding the factors to the collection of attributes are very similar, with $\pm 1\,\%$ difference only.

TABLE II
CLASSIFICATION ACCURACY INCREASE FOR DATASETS FROM TAB. I, FOR $w = 1$ (LEFT TABLE) AND $w = 0$ (RIGHT TABLE) IN (2)

training % / *testing %*	*ID3*	*C4.5*	*IB1*
breast-cancer	1.020	1.031	1.020
	1.159	0.989	0.970
kr-vs-kp	1.000	0.998	1.000
	0.993	0.994	1.017
mushroom	1.000	1.000	1.000
	1.000	1.000	1.000
tic-tac-toe	1.000	1.028	1.000
	1.123	1.092	1.000
vote	1.000	0.998	1.000
	0.993	0.994	1.005
average	1.004	1.011	1.004
	1.054	1.014	0.998

training % / *testing %*	*ID3*	*C4.5*	*IB1*
breast-cancer	1.020	1.047	1.020
	1.153	1.035	0.951
kr-vs-kp	1.000	1.000	1.000
	1.000	0.998	1.083
mushroom	1.000	1.000	1.000
	1.000	1.000	1.000
tic-tac-toe	1.000	1.033	1.000
	1.157	1.138	1.213
vote	1.000	1.000	1.000
	1.017	1.007	1.033
average	1.004	1.016	1.004
	1.065	1.036	1.056

ACKNOWLEDGMENT

The research was supported by grant no. P202/10/P360 of the Czech Science Foundation.

REFERENCES

[1] R. Belohlavek and V. Vychodil: Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. System Sci* **76**(1)(2010), 3-20.

[2] L. Breiman, J. H. Friedman, R. Olshen and C. J. Stone: *Classification and Regression Trees.* Chapman & Hall, NY, 1984.

[3] C. Carpineto and G. Romano: *Concept Data Analysis. Theory and Applications.* J. Wiley, 2004.

[4] B. Ganter and R. Wille: *Formal Concept Analysis. Mathematical Foundations.* Springer, Berlin, 1999.

[5] Harman H. H.: *Modern Factor Analysis, 2nd Ed.* The Univ. Chicago Press, 1970.

[6] K. H. Kim: *Boolean Matrix Theory and Applications.* M. Dekker, 1982.

[7] R. S. Michalski: A theory and methodology of inductive learning. *Artificial Intelligence* **20**(1983), 111–116.

[8] P. M. Murphy and M. J. Pazzani: ID2-of-3: constructive induction of M-of-N concepts for discriminators in decision trees. In Proc. of the Eight Int. Workshop on Machine Learning, 1991, 183–187.

[9] S. K. Murthy: Automatic construction of decision trees from data. *Data Mining and Knowledge Discovery* 2, 1998, 345–389.

[10] S. K. Murthy, S. Kasif and S. Salzberg: A system for induction of oblique decision trees. *J. of Artificial Intelligence Research* **2**(1994), 1–33.

[11] D. J. Newman, S. Hettich, C. L. Blake and C. J. Merz: *UCI Repository of machine learning databases*, [`http://www.ics.uci.edu/~mlearn/MLRepository.html`], University of California, Dept. of Information and Computer Science, 1998.

[12] G. Pagallo and D. Haussler: Boolean feature discovery in empirical learning. *Machine Learning* **5**(1)(1990), 71–100.

[13] J. R. Quinlan: *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[14] P. N. Tan, M. Steinbach and V. Kumar: *Introduction to Data Mining.* Addison Wesley, Boston, MA, 2006.

[15] I. H. Witten and E. Frank: *Data Mining: Practical machine learning tools and techniques, 2nd Edition.* Morgan Kaufmann, San Francisco, 2005.