Open access • Journal Article • DOI:10.1016/S0169-023X(02)00057-5

Computing iceberg concept lattices with TITANIC — Source link

Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier ...+1 more authors

Institutions: Karlsruhe Institute of Technology, French Institute for Research in Computer Science and Automation, Blaise Pascal University, University of Nice Sophia Antipolis ...+1 more institutions

Published on: 01 Aug 2002 - Data and Knowledge Engineering

Topics: Conceptual clustering, Knowledge extraction, Formal concept analysis, Association rule learning and Closure (topology)

Computing iceberg concept lattices with Titanic

Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, Lotfi Lakhal

ELSEVIER

Data & Knowledge Engineering 42 (2002) 189–222

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Computing iceberg concept lattices with TITANIC

Gerd Stumme [a,*], Rafik Taouil [b], Yves Bastide [c], Nicolas Pasquier [d], Lotfi Lakhal [e]

[a] *Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), Universität Karlsruhe (TH), D-76128 Karlsruhe, Germany*

[b] *INRIA Lorraine, LORIA, BP 239, F-54506 Vandœuvre-lès-Nancy, France*

[c] *Laboratoire d'Informatique (LIMOS), Université Blaise Pascal, Complexe Scientifique des Cézeaux, 24 Avenue des Landais, F-63177 Aubière Cedex, France*

[d] *Université de Nice, I3S—CNRS UPRESA 6070-UNSA, Les Algorithmes—Euclide B, 2000 route des Lucioles, BP 121, F-06903 Sophia Antipolis, France*

[e] *LIM, CNRS FRE-2246, Université de la Méditerranée, Case 90, 163 Avenue de Luminy, F-13288 Marseille Cedex 9, France*

Abstract

We introduce the notion of *iceberg concept lattices* and show their use in knowledge discovery in databases. Iceberg lattices are a conceptual clustering method, which is well suited for analyzing very large databases. They also serve as a condensed representation of frequent itemsets, as starting point for computing bases of association rules, and as a visualization method for association rules. Iceberg concept lattices are based on the theory of Formal Concept Analysis, a mathematical theory with applications in data analysis, information retrieval, and knowledge discovery. We present a new algorithm called TITANIC for computing (iceberg) concept lattices. It is based on data mining techniques with a level-wise approach. In fact, TITANIC can be used for a more general problem: Computing arbitrary closure systems when the closure operator comes along with a so-called weight function. The use of weight functions for computing closure systems has not been discussed in the literature up to now. Applications providing such a weight function include association rule mining, functional dependencies in databases, conceptual clustering, and ontology engineering. The algorithm is experimentally evaluated and compared with Ganter's Next-Closure algorithm. The evaluation shows an important gain in efficiency, especially for weakly correlated data. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Knowledge discovery; Database analysis; Formal concept analysis; Closure systems; Lattices; Algorithms

[*] Corresponding author.

E-mail addresses: stumme@aifb.uni-karlsruhe.de (G. Stumme), rafik.taouil@loria.fr (R. Taouil), bastide@libd2.univ-bpclermont.fr (Y. Bastide), pasquier@mezzo.unice.fr (N. Pasquier), lotfi.lakhal@lim.univ-mrs.fr (L. Lakhal).

PII: S0169-023X(02)00057-5

1. Introduction

Concept Lattices are used to represent conceptual hierarchies which are inherent in data. They are the core of the mathematical theory of Formal Concept Analysis (FCA). Introduced in the early 1980s as a formalization of the concept of 'concept' [46], FCA has over the years grown to a powerful theory for data analysis, information retrieval, and knowledge discovery [43]. In artificial intelligence (AI), FCA is used as a knowledge representation mechanism [48] and as conceptual clustering method [8,27,37]. In database theory, FCA has been extensively used for class hierarchy design and management [10,13,28,35,45,50]. Its usefulness for the analysis of data stored in relational databases has been demonstrated with the commercially used management system TOSCANA for Conceptual Information Systems [44].

A current research domain common to both the AI and the database community is knowledge discovery in databases (KDD). Here FCA has been used as a formal framework for implication and association rules discovery and reduction [31,41] and for improving the response times of algorithms for mining association rules [31,32]. The interaction of FCA and KDD in general has been discussed in [42] and [16].

In this paper we show that, vice versa, FCA can also benefit from ideas used for mining association rules: Computing concept lattices is an important issue, investigated for long years [11,14,28,29,50]. We address the problem of computing concept lattices from a data mining viewpoint by using a level-wise approach [2,25]; and provide a new, efficient algorithm called Titanic. In fact, Titanic can be used for a more general problem: Computing arbitrary closure systems when the closure operator comes along with a so-called weight function. The use of weight functions for computing closure systems has not been discussed in the literature up to now. Weight functions appear naturally in a variety of applications, include association rule mining, functional dependencies in databases, conceptual clustering, ontology learning, transformation of class hierarchies in object-oriented languages, and configuration space analysis in software reengineering.

We also introduce the notion of *iceberg concept lattices*. Iceberg concept lattices show only the top-most part of a concept lattice. Iceberg concept lattices have different uses in KDD: as conceptual clustering tool, as a visualization method, especially for *very large* databases, as a condensed representation of frequent itemsets, as a base of association rules, and as a visualization tool for association rules.

In Section 2, we recall the basics of FCA. In Section 3, we introduce iceberg concept lattices and explain their use as conceptual clustering method by an example. Section 4 provides the theoretical foundation and gives a formal statement of the generalized problem of computing closure systems using a weight function. The problem is split in several subtasks in Section 5, and turned into pseudo-code in Section 6. In Section 7, we apply the algorithm to concept lattices and iceberg concept lattices, and provide examples. Section 8 lists some typical applications. In Section 9, we provide a complexity discussion and an experimental evaluation. Section 10 concludes the article.

This article consolidates research presented in the workshop papers [39,40].

2. Formal concept analysis

Since concepts are necessary for expressing human knowledge, any knowledge management process benefits from a comprehensive formalization of concepts. FCA offers such a formalization by mathematizing the concept of 'concept' as a unit of thought constituted of two parts: its extension and its intension [12,46]. This understanding of 'concept' is first mentioned explicitly in the Logic of Port Royal [3] and has been established in the international standard ISO 704.

We recall the basics of FCA as far as they are needed for this paper. The definitions and theorems in this subsection are quoted from [46]. A more extensive overview is given in [12].

To allow a mathematical description of extensions and intensions, FCA starts with a (*formal*) *context*. [1]

Definition 1. A formal context is a triple $\mathbb{K} := (G, M, I)$ where G and M are sets and $I \subseteq G \times M$ is a binary relation. The elements of G are called objects and the elements of M attributes. The inclusion $(g, m) \in I$ is read "object g has attribute m". For $A \subseteq G$, we define

$$A' := \{m \in M | \forall g \in A : (g, m) \in I\};$$

and for $B \subseteq M$, we define dually

$$B' := \{g \in G | \forall m \in B : (g, m) \in I\}.$$

We assume, in this article, that all sets are finite, especially G and M.

Lemma 1. *Let (G, M, I) be a context, $A_1, A_2 \subseteq G$ sets of objects, and $B_1, B_2 \subseteq M$ sets of attributes. Then the following holds*:

$$A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1' \tag{1}$$

$$B_1 \subseteq B_2 \Rightarrow B_2' \subseteq B_1' \tag{1a}$$

$$A \subseteq A'' \tag{2}$$

$$B \subseteq B'' \tag{2a}$$

$$A' = A''' \tag{3}$$

$$B' = B''' \tag{3a}$$

$$A \subseteq B' \iff B \subseteq A' \iff A \times B \subseteq I. \tag{4}$$

Definition 2. A formal concept is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. (This is equivalent to $A \subseteq G$ and $B \subseteq M$ being maximal with $A \times B \subseteq I$.) A is called extent and B is called intent of the concept.

[1] The notion of context has also been used in many other AI applications. See http://extractor.iit.nrc.ca/bibliographies/context sensitive for references. In this paper, we always refer to the notion developed in FCA.

The set $\underline{\mathfrak{B}}(\mathbb{K})$ of all concepts of a formal context $\mathbb{K}$ together with the partial order $(A_1,B_1)\leqslant(A_2,B_2)$: $\Longleftrightarrow A_1\subseteq A_2$ (which is equivalent to $B_1\supseteq B_2$) is called concept lattice of $\mathbb{K}$.

The following lemma shows, together with Lemma 1 (3a), that a concept lattice can be derived from the set of its concept intents.

Lemma 2. *Let $\mathbb{K}:=(G,M,I)$ be a formal context. Then*

$$\underline{\mathfrak{B}}(\mathbb{K})=\{(B',B'')|B\subseteq M\}.$$

The fundamental theorem of FCA [46] shows that each concept lattice is a complete lattice, and that the set of its intents is a closure system (see Section 5):

Theorem 3 (Fundamental Theorem of FCA). *Let $\mathbb{K}:=(G,M,I)$ be a formal context. Then $\underline{\mathfrak{B}}(\mathbb{K})$ is a complete lattice in which infima and suprema can be described as follows*:

$$\bigwedge_{j\in J}(A_j,B_j)=\left(\bigcap_{j\in J}A_j,\left(\bigcup_{j\in J}B_j\right)''\right),\quad \bigvee_{j\in J}(A_j,B_j)=\left(\left(\bigcup_{j\in J}A_j\right)'',\bigcap_{j\in J}B_j\right)$$

Conversely, if L is a complete lattice then $L\cong\underline{\mathfrak{B}}(\mathbb{K})$ if and only if there are mappings $\gamma:G\to L$ and $\mu:M\to L$ such that $\gamma(G)$ is supremum-dense in L, $\mu(M)$ is infimum-dense in L, and $(g,m)\in L$ is equivalent to $\gamma(g)\leqslant\mu(m)$, for all $g\in G$ and $m\in M$. In particular, $L\cong\underline{\mathfrak{B}}(L,L,\leqslant)$.

Example. As running example, we use the Mushroom database from the *UCI KDD Archive* (http://kdd.ics.uci.edu/). It consists of a database with 8416 objects (Mushrooms) and 22 (nominally valued) attributes. We obtain a formal context by creating one (Boolean) attribute for each of the 80 possible values of the 22 database attributes. The resulting formal context has thus 8416 objects and 80 attributes. In order to explain FCA by a small example, we restrict ourselves first to a very limited sub-context, namely the first 10 objects, and 13 attributes. This restricted formal context is shown in Fig. 1. A line diagram of its concept lattice is shown in Fig. 2.

In the line diagram, the name of an object g is always attached to the circle representing the smallest concept with g in its extent; dually, the name of an attribute m is always attached to the circle representing the largest concept with m in its intent. This allows us to read the context relation from the diagram because an object g has an attribute m if and only if there is an ascending path from the circle labeled by g to the circle labeled by m. The extent of a concept consists of all objects whose labels are below in the hierarchy, and the intent consists of all attributes attached to concepts above in the hierarchy. For example, the concept without label in the middle of the diagram has {Mushroom 3, Mushroom 4} as extent, and {edible, cap surface: fibrous, cap shape:flat} as intent.

For X, $Y\subseteq M$, we say that the *implication* $X\Rightarrow Y$ holds in the context, if each object having all attributes in X also has all attributes in Y (i.e., an implication is an association rule [2] with 100%

[2] An association rule is a pair $X\to Y$ with $X,Y\subseteq M$. Its support is defined by $\mathrm{supp}(X\to Y):=(|(X\cup Y)'|)/|G|$, and its *confidence* by $\mathrm{conf}(X\to Y):=(|(X\cup Y)'|)/|X'|)$. See [1].

	edible	poisonous	cap shape: convex	cap shape: flat	cap surface: fibrous	cap surface: scaly	cap surface: smooth	cap color: brown	cap color: buff	cap color: gray	cap color: red	cap color: white	cap color: yellow
Mushroom 1	×		×				×					×	
Mushroom 2	×		×		×					×			
Mushroom 3	×			×	×							×	
Mushroom 4	×			×	×			×					
Mushroom 5	×		×		×					×			
Mushroom 6	×		×			×					×		
Mushroom 7		×		×	×					×			
Mushroom 8		×		×	×					×			
Mushroom 9		×		×	×								×
Mushroom 10		×		×			×		×				

Fig. 1. Formal context about Mushrooms.

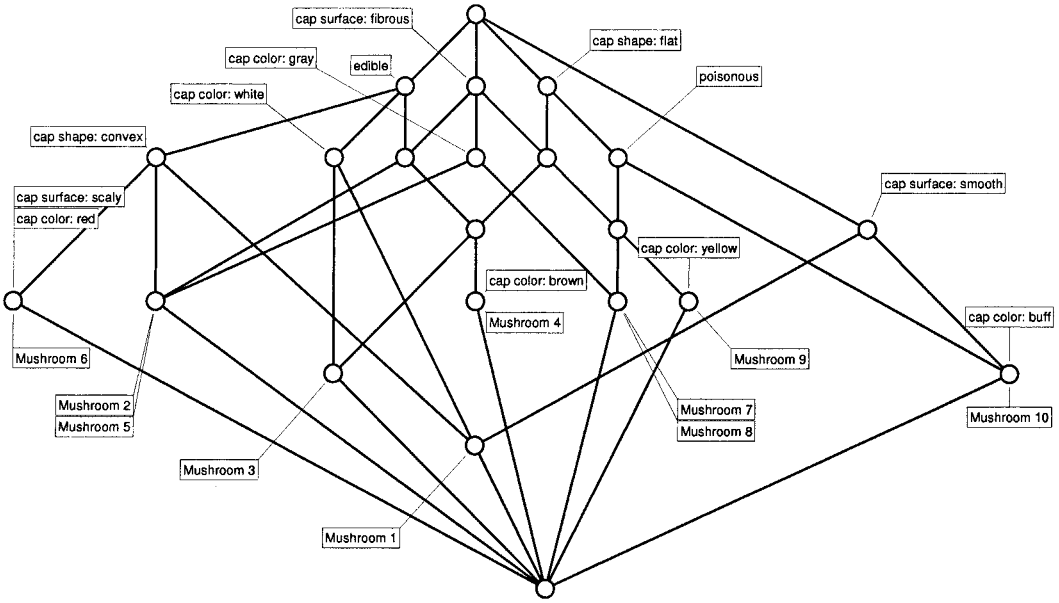

Fig. 2. The concept lattice of the context of Fig. 1.

confidence). For instance, the implication {cap shape: flat, cap surface:smooth} ⇒ {cap color: buff, poisonous} holds in the context. (Of course it may not hold any longer when we enlarge the set of objects under consideration.)

Implications can be read directly in the line diagram: the largest concept having both 'cap shape: flat' and 'cap surface: smooth' in its intent is just the concept labeled by 'cap color: buff'—which on its turn lies below the concept labeled by 'poisonous'. In the next section is discussed how also association rules with less than 100% confidence can by visualized in the line diagram.

Beside association rule mining, FCA has been applied in a wide range of application domains, including medicine, psychology, social sciences, linguistics, information sciences, machine and civil engineering etc. (cf. [43]). Over all, FCA has been used in more than 200 projects, both on the scientific and the commercial level. For instance, FCA has been applied for analyzing data of children with diabetes [34], for developing qualitative theories in music esthetics [23], for managing emails [9], for database marketing [16], and for an IT security management system [7].

3. Iceberg concept lattices

The previous example was unsatisfying insofar as it was restricted to a very small and—more important—arbitrarily chosen set of objects. On the other hand, this restriction allowed us to display the entire concept lattice. In the worst case, the size of a concept lattices is exponentially in the size of the context. Hence for most applications one has to consider strategies (other than arbitrarily reducing the context) for dealing with such large concept lattices.

In this paper, we present an approach based on frequent itemsets as known from data mining [1]: Our iceberg concept lattices will consist only of the top-most concepts of the concept lattice. These are the concepts which provide the most global structuring of the domain.

Definition 3. Let $B \subseteq M$, and let minsupp $\in [0,1]$. The support count of the attribute set (also called itemset) B in $\mathbb{K}$ is $\mathrm{supp}(B) := |B'|/|G|$. B is said to be a frequent attribute set if $\mathrm{supp}(B) \geqslant$ minsupp.

A concept is called frequent concept if its intent is frequent. The set of all frequent concepts of a context $\mathbb{K}$ is called the iceberg concept lattice of the context $\mathbb{K}$.

Because the support function is monotonously decreasing (i.e., $B_1 \subseteq B_2 \Rightarrow \mathrm{supp}(B_1) \geqslant \mathrm{supp}(B_2)$), the iceberg concept lattice is an order filter of the whole concept lattice, and thus in general only a join-semi-lattice. But when we add a new bottom element, it becomes a lattice again. This makes it possible to apply the same algorithm (which will be introduced in the following sections) for computing concept lattices and iceberg concept lattices. But before talking about their computation, let us have a closer look to iceberg concept lattices:

Example. Now we consider the whole Mushroom database. Its concept lattice consists of 32,086 concepts, hence is by far too large to be displayed. But for a first glance, it is sufficient to see its top-most part: Fig. 3 shows the Mushroom iceberg concept lattice for a minimum support of 85%.

In the diagram one can clearly see that all Mushrooms in the database have the attribute ‘veil type: partial’. Furthermore the diagram tells us that the three next-most attributes are: ‘veil color: white’ (with 97.62% support), ‘gill attachment: free’ (97.43%), and ‘ring number: one’ (92.30%). There is no other attribute having a support higher than 85%. But even the combination of all these four concepts is frequent (with respect to our threshold of 85%): 89.92% of all Mushrooms in our database have one ring, a white partial veil, and free gills. This concept with a quite complex description contains more objects than the concept described by the fifth-most attribute, which has a support below our threshold of 85%, since it is not displayed in the diagram.

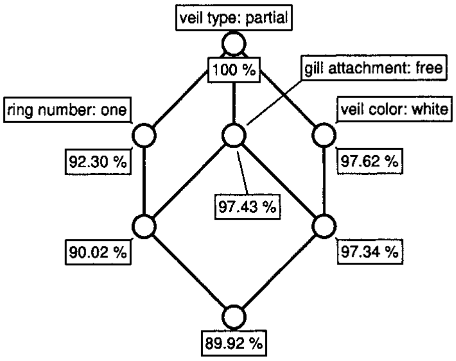

Fig. 3. Iceberg concept lattice of the Mushroom database with minsupp = 85%.

In the diagram, we can detect the implication

$$\{\text{ring number: one, veil color: white}\} \Rightarrow \{\text{gill attachment: free}\}.$$

It is indicated by the fact that there is no concept having ‘ring number: one’ and ‘veil color: white’ (and ‘veil type: partial’) in its intent, but not ‘gill attachment: free’. This implication has a support of 89.92% (and as it is an implication, a confidence of 100%). Unlike the implications in Example 1 (which hold for the ten objects under consideration only), this implication is globally valid, i.e., it does not change when we consider a different minimum support.

If we want to see more details, we have to decrease the minimum support. Fig. 4 shows the Mushroom iceberg concept lattice for a minimum support of 70%. One observes that, of course, its top-most part is just the iceberg lattice for minsupp = 85%. Additionally, we obtain five new concepts, having the possible combinations of the next-most attribute ‘gill spacing: close’ (having

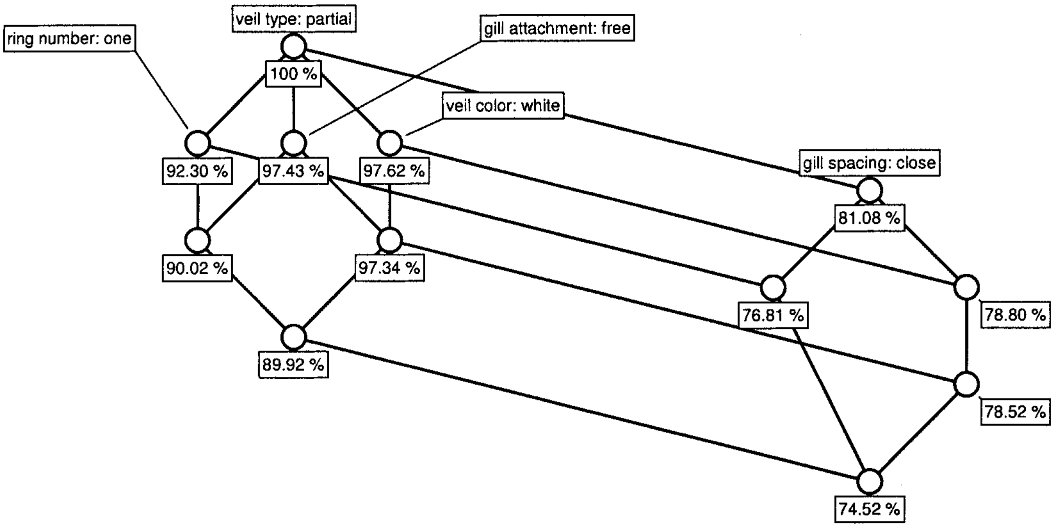

Fig. 4. Iceberg concept lattice of the Mushroom database with minsupp = 70%.

support 81.08%) with the previous four attributes. The fact that the combination {veil type: partial, gill attachment: free, gill spacing: close} is not realized as a concept intent indicates another implication:

$$\{\text{gill attachment: free, gill spacing: close}\} \Rightarrow \{\text{veil color: white}\} \qquad (*)$$

This implication has 78.52% support (the support of the most general concept having all three attributes in its intent) and— being an implication—100% confidence.

By further decreasing the minimum support, we discover more and more details. Fig. 5 shows the MUSHROOMS iceberg concept lattice for a minimum support of 55%. It shows four more partial copies of the 85% iceberg lattice, and three new, single concepts.

The MUSHROOMS example shows that iceberg concept lattices are suitable especially for strongly correlated data. In Table 1, the size of the iceberg lattice (i.e., the number of all frequent closed itemsets) is compared with the number of all frequent itemsets. It shows for instance, that, for the minimum support of 55%, only 32 frequent closed itemsets are needed to provide all information about the support of all 116 frequent itemsets one obtains for the same threshold.

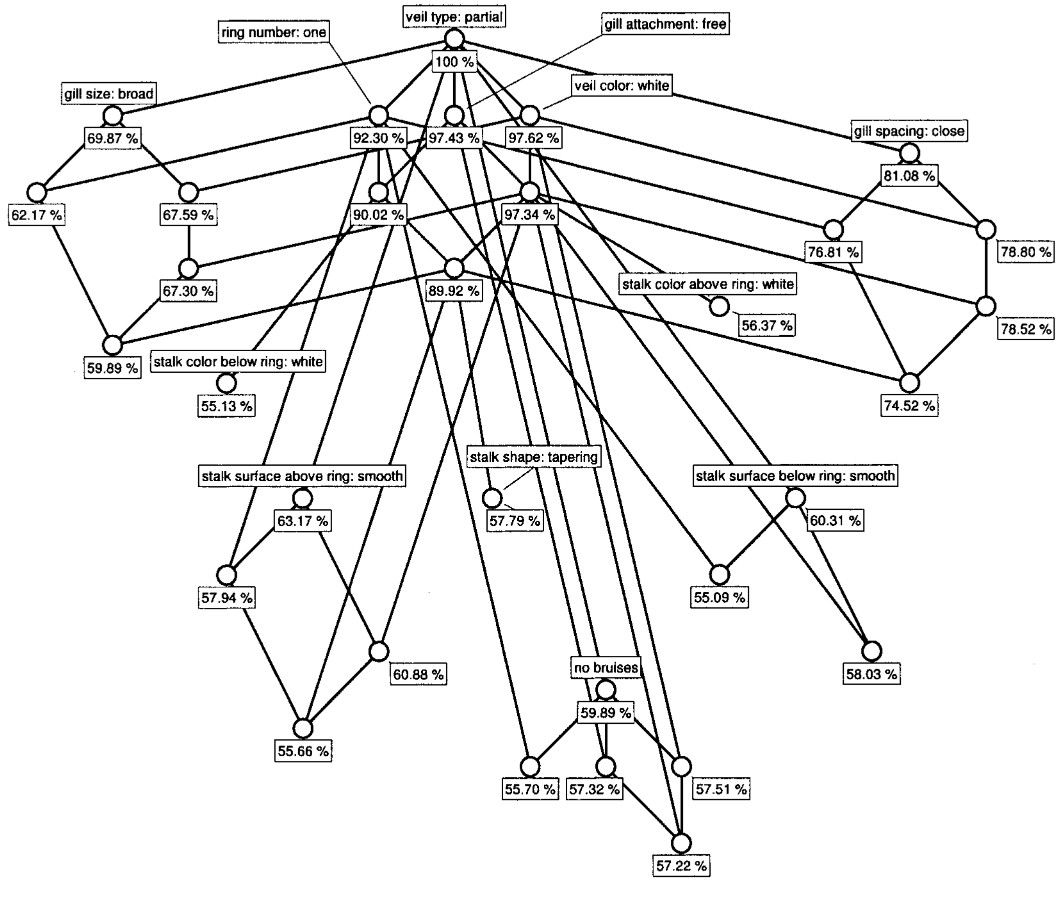

Fig. 5. Iceberg concept lattice of the MUSHROOM database with minsupp = 55%.

Table 1
Number of frequent closed itemsets and frequent itemsets for the Mushrooms example

Minsupp (%)	# Frequent closed itemsets	# Frequent itemsets
85	7	8
70	12	32
55	32	116
0	32.086	2^{80}

The observation that the top-most part of the iceberg lattice appears partially again in combination with other attributes can be used for an alternative visualization: Fig. 6 shows the iceberg concept lattice as a *nested line diagram*. The diagram provides exactly the same information than Fig. 5, but in a more structured way.

Each of the 'satellites' contains a partial copy of the top-most iceberg lattice. Only those concepts are copied which are, together with the new attribute(s), still frequent. The lines of the outer diagram have to be read as a bundle of parallel lines, linking corresponding concepts. For instance, the concept on the right side of the diagram labeled by '78.80%' is not only an immediate subconcept of the one labeled by '81.08%, but also of the one labeled by '97.62%'.

The empty circles indicate *unrealized concepts*: They are still frequent, but all objects in an unrealized concept share at least one more attribute. For instance, the unrealized concept on the right side left of the concept labeled by '78.80%' has as intent {gill spacing: close, gill attachment: free, veil type: partial}. But implication (∗) tells us that all objects having these attributes also have the attribute 'veil color: white'. Therefore, 'veil color: white' has to be in each realized concept which contains the three other attributes. The largest of them is just the first realized concept below: the one with 78.52% support. This way, each unrealized concept indicates an implication: the attributes of its intent always imply all attributes in the intent of its largest realized subconcept. For instance, the two unrealized concepts below the attribute 'no bruises' indicate the implications

$$\{\text{no bruises, gill attachment: free}\} \Rightarrow \{\text{veil color: white}\}$$
$$\{\text{no bruises, veil color: white}\} \Rightarrow \{\text{gill attachment: free}\}$$

respectively, each having 57.22% support.

For attributes which are labeled at concepts having no subconcepts in the diagram, we cannot decide whether they are part of interesting implications. For instance, the diagram does not show whether there is an implication having 'stalk color below ring: white' in its premise or conclusion (other than the trivial implication {stalk color below ring: white} ⇒ {veil type: partial}). If there are any such rules, then their support is below the actual minimum support of 55%. In order to study them, the threshold has to be decreased further.

In the way nested line diagrams are introduced in [47], the attributes are grouped manually according to their semantics. Related attributes are grouped together. This usually involves a human expert to decide which attributes are related. The support function, on the other hand, allows an automatic grouping: In Fig. 6, the inner diagram contains the top-most attributes, the outer diagram the next-most attributes. The resulting diagram shows the most important

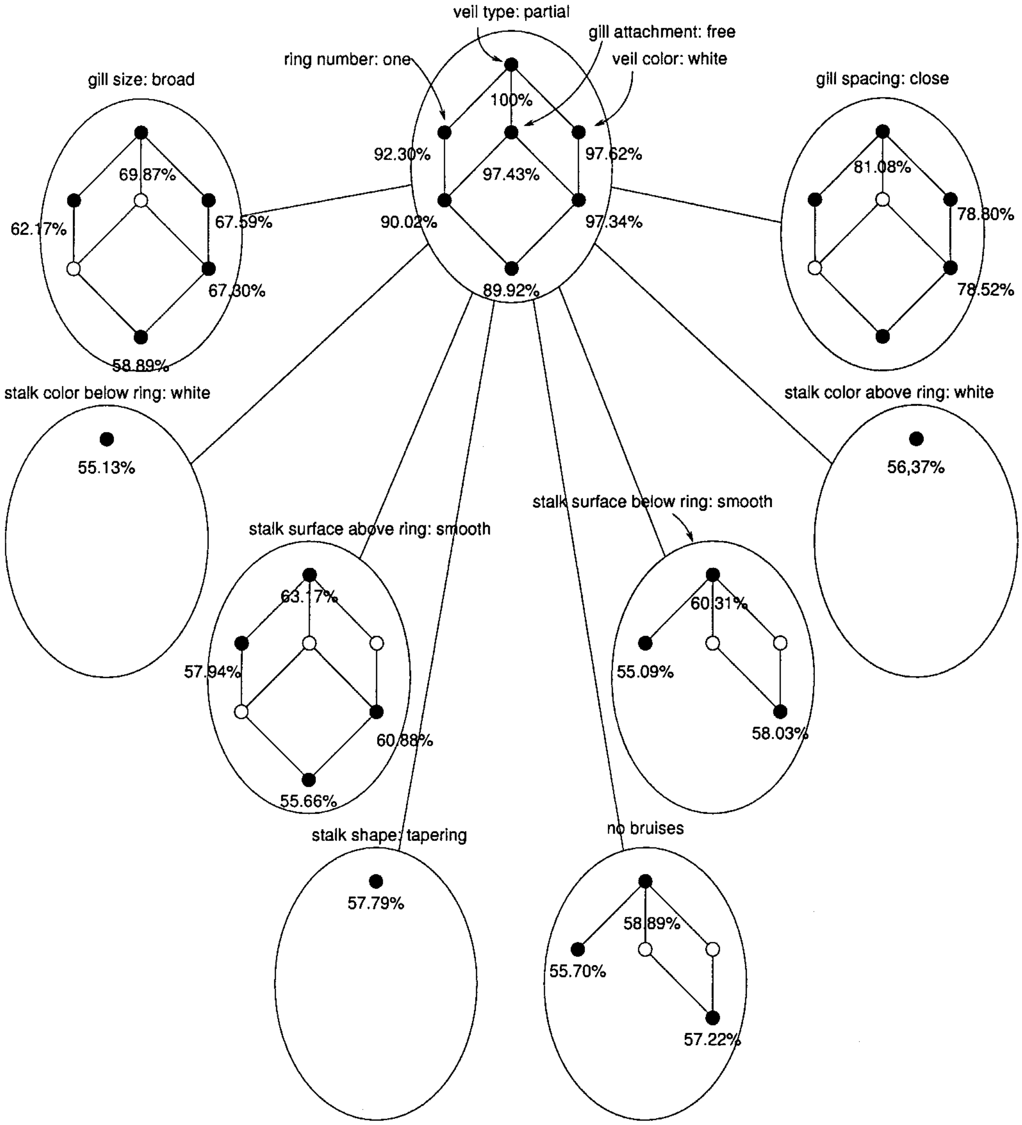

Fig. 6. Nested line diagram of the iceberg concept lattice in Fig. 5.

attributes for structuring the domain. The knowledge engineer only has to fix the minimum support thresholds for the different layers.

Observe that the iceberg concept lattices in this example are used for *conceptual clustering*, or *un-supervised learning*. Our aim was to gain new insights about the Mushrooms in the database, independent from a specific purpose. In particular, the aim was not to learn how to distinguish between poisonous and edible Mushrooms. The question if and how iceberg concept lattices can be used in such a *supervised learning* scenario is an interesting open problem.

In general, cluster analysis comprises a set of unsupervised machine learning techniques which split sets of objects into clusters (subsets) such that objects within a cluster are as similar as possible while objects from different clusters are as different as possible. Conceptual clustering techniques additionally aim at determining not only clusters, i.e., concept extensions, but to provide at the same time intensional descriptions of these extensions [26,49]. This aim fits well with the understanding of concepts formalized in FCA. Therefore FCA was considered as a framework for conceptual clustering from the early 1990s on [8,27,37].

Compared to 'usual' clustering, conceptual clustering techniques pay their added value (the intensional description) with increased computation time. In FCA, there exist basically three ways to overcome this problem: local focusing (e.g., [8]), vertical reduction by conceptual scaling [12], and horizontal reduction. Iceberg concept lattices are a horizontal approach to reduce the amount of information (and the computation time) of a concept lattice. In comparison to other conceptual clustering approaches, iceberg concept lattices have structural properties which can be stated explicitly: they do not depend on diverse parameters (except the minimum support threshold) whose semantics are often difficult to interpret, nor on the order in which the input is presented to the algorithm, nor on any particularities of the implementation. Another distinction to other hierarchical clustering results is that they allow for multiple hierarchies (and not only for trees), so that all potentially interesting specialization paths are contained in the resulting hierarchy.

Up to now, we have discussed the use of iceberg concept lattices as a conceptual clustering technique, equipped with a visualization method, which is very well suited especially for analyzing very large databases containing strongly correlated data. Now we briefly discuss some more uses of iceberg concept lattices in KDD.

3.1. A condensed representation of frequent itemsets

The computation of frequent attribute sets (itemsets) is the first (and most expensive) step in the computation of association rules. One reason is that one needs to count the support for each itemset. By using the fact that $\mathrm{supp}(B) = \mathrm{supp}(B'')$, for $B \subseteq M$, we can derive the supports of all itemsets from the supports of the frequent concept intents only. In strongly correlated data, only relatively few of the frequent itemsets are also concept intents. Hence only few support counts have to be effected in the database. This is used for the Pascal algorithm [5] which is related to Titanic, and which efficiently computes frequent itemsets.

3.2. A starting point for computing bases of association rules

One problem in mining association rules is the large number of rules which are usually returned. In [4] and [41], different bases for association rules are introduced, which prune redundant rules, but from which all valid rules can still be derived. The computation of the bases does not require all frequent itemsets, but only frequent concept intents.

3.3. A visualizing technique for association rules

We have already discussed how implications (i.e., association rules with 100% confidence) can be read from the line diagram. The Luxenburger basis for approximate association rules (i.e.,

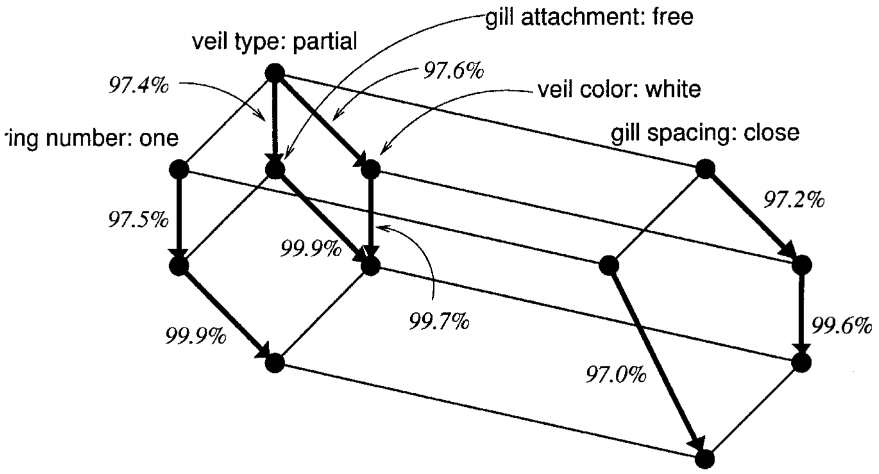

Fig. 7. Visualization of the Luxenburger basis for minsupp = 70% and minconf = 95%.

association rules with less than 100% confidence), which is presented in [41], can also be visualized directly in the line diagram of an iceberg concept lattice. The Luxenburger basis is derived from [22]. It contains only those rules $B_1 \to B_2$ where B_1 and B_2 are frequent concept intents and where the concept (B_1', B_1) is an immediate subconcept of (B_2', B_2). Hence there corresponds to each approximate rule in the Luxenburger base exactly one edge in the line diagram. Fig. 7 visualizes all rules in the Luxenburger basis for minsupp = 70% and minconf = 95%. For instance, the rightmost arrow stands for the association rule {veil color: white, gill spacing: close} → {gill attachment: free}, which holds with a confidence of 99.6%. Its support is the support of the concept the arrow is pointing to: 78.52%, as shown in Fig. 4. Edges without label indicate that the confidence of the rule is below the minimum confidence threshold. The visualization technique is described in more detail in [41]. In comparison with other visualization techniques for association rules (as for instance implemented in the IBM Intelligent Miner), the visualization of the Luxenburger basis within the iceberg concept lattice benefits of the smaller number of rules to be represented (without loss of information!), and of the presence of a 'reading direction' provided by the concept hierarchy.

4. Computing closure systems: the problem

Instead of giving an algorithm for computing (iceberg) concept lattices, we provide an algorithm for a more general task: computing closure systems using a weight function. The reason is that closure systems are important in a variety of applications. Some example applications are given in Section 8. In this section, we formally state the problem, and in the next section, we present our approach. Its efficiency is discussed in Section 9.

First, we recall the definition of closure systems:

Definition 4. A closure system on a set M is a subset $\mathscr{H}$ of the powerset $\mathfrak{P}(M)$ of M which contains the set M and which is closed under arbitrary intersections. A closure operator on a set M is a function $h : \mathfrak{P}(M) \to \mathfrak{P}(M)$ which is

- *extensive*: $X \subseteq h(X)$
- *monotonous*: $X \subseteq Y \Rightarrow h(X) \subseteq h(Y)$
- *and idempotent*: $h(h(X)) = h(X)$.

It is well-known that closure operators and closure systems are equivalent: For each closure operator h, the set $\mathscr{H}_h := \{X \subseteq M | h(X) = X\}$ is a closure system on M; for each closure system $\mathscr{H}$ the function $h_{\mathscr{H}} : \mathfrak{P}(M) \to \mathfrak{P}(M)$ with $X \mapsto \bigcap_{H \in \mathscr{H}, H \supseteq X} H$ is a closure operator; and the following two equations hold: $\mathscr{H}_{h_{\mathscr{H}}} = \mathscr{H}$ and $h_{\mathscr{H}_h} = h$.

Lemma 1 and Theorem 3 show that the set of all intents of a context (G, M, I) is a closure system on M, and that $B \mapsto B''$ is the corresponding closure operator. Thus computing concept lattices is a special case of the following, more general task.

Let h be a closure operator on a finite set M. The task is to determine efficiently the closure system $\mathscr{H}_h$ related to the closure operator h when there exists a *weight function* compatible with the closure operator:

Definition 5. A weight function on $\mathfrak{P}(M)$ is a function $s : \mathfrak{P}(M) \to P$ from the powerset of M to a totally ordered set $(P, \leqslant)$ having a largest element $s_{\max}$. For a set $X \subseteq M$, $s(X)$ is called the weight of X. The weight function is compatible with a closure operator h if

(i) $X \subseteq Y \Rightarrow s(X) \geqslant s(Y)$,
(ii) $h(X) = h(Y) \Rightarrow s(X) = s(Y)$,
(iii) $X \subseteq Y \wedge s(X) = s(Y) \Rightarrow h(X) = h(Y)$.

Remark. In the sequel, we will consider $(P, \leqslant)$ to be the interval [0,1] in the real numbers, but the theory presented in this paper can be applied to arbitrary totally ordered sets.

Remark. If $X \subseteq Y \Rightarrow s(X) \leqslant s(Y)$ holds instead of (i) (as, e.g., for functional dependencies), then all 'min' in the sequel have to be replaced by 'max'.

Now we can formally state the problem:

Problem. Let h be a closure operator on a finite set M, and let s be a compatible weight function. Determine the closure system $\mathscr{H}_h$ related to the closure operator h by using the weight function s.

5. Computing closure systems based on weights

We discuss the problem of computing the closure system by using a weight function in three parts:

(1) How can we compute the closure of a given set using the weight function only, and not the closure operator?
(2) How can we compute the closure system by computing as few closures as possible?
(3) Since the weight function is usually not stored explicitly, how can we derive the weights of as many sets as possible from the weights already computed?

Questions 2 and 3 are not independent from each other. Hence we will not provide an optimal answer for each of them, but one which improves the overall benefit.

5.1. Weight-based computation of closures

We use the constraints on the function s for determining the closure of a set by comparing its weight with the weights of its immediate supersets.

Proposition 4. *Let $X \subseteq M$. Then*

$$h(X) = X \cup \{m \in M \setminus X | s(X) = s(X \cup \{m\})\}.$$

Proof. "$\subseteq$": Suppose that there exists $m \in h(X) \setminus X$ with $s(X) \neq s(X \cup \{m\})$. Then $h(X) \neq h(X \cup \{m\})$ by condition 2 of Definition 5. Hence $m \notin h(X)$. Contradiction.

"$\supseteq$": Let $m \in M \setminus X$ with $s(X) = s(X \cup \{m\})$. Then $h(X) = h(X \cup \{m\})$ by condition 3 of Definition 5. Hence $m \in h((X \cup \{m\})) = h(X)$. □

Hence if we know the weights of all sets, then we can compute the closure operator (→ Algorithm 3, steps 3–7). [3] In Section 5.2 we discuss for which sets it is necessary to compute the closure in order to obtain all closed sets. In Section 6 we discuss how the weights needed for those computations can be determined.

5.2. A level-wise approach for computing all closed sets

One can now compute the closure system $\mathscr{H}$ by applying Proposition 4 to all subsets X of M. But this is not efficient, since many closed sets will be determined several times.

Definition 6. We define an equivalence relation θ on the powerset $\mathfrak{P}(M)$ of M by $(X, Y) \in \theta :\iff h(X) = h(Y)$, for $X,\ Y \subseteq M$. The equivalence class of X is given by $[X] := \{Y \subseteq M | (X, Y) \in \theta\}$.

If we knew the equivalence relation θ in advance, it would be sufficient to compute the closure for one set of each equivalence class only. But since we have to determine the relation during the computation, we have to consider more than one element of each class in general. As known from algorithms for mining association rules, we will use a level-wise approach.

[3] In this section, we give some references to the algorithms in the following section. These references can be skipped at the first reading.

Definition 7. A k-set is a subset X of M with $|X| = k$. it For $\mathscr{X} \subseteq \mathfrak{P}(M)$, we define $\mathscr{X}_k := \{X \in \mathscr{X} | X \text{ is } k\text{-set}\}$. For $\mathscr{X} = \mathfrak{P}(M)$, we also write $\mathfrak{P}_k(M)$ for $\mathscr{X}_k$.

At the kth iteration, the weights of all k-sets which remained from the pruning strategy described below are determined; and the closures of all $(k-1)$-sets which passed the pruning in the $(k-1)$th iteration are computed.

The first sets of an equivalence class that we reach using such a level-wise approach are the minimal sets in the class.

Definition 8. A set $X \subseteq M$ is a key set (or minimal generator) if X is minimal (with respect to set inclusion) in $[X]$. The set of all key sets is denoted by $\mathscr{K}$.

We have $\mathscr{H} = \{h(X) | X \in \mathscr{K}\}$, because there is at least one key set in each equivalence class of θ. Hence it is sufficient to compute the closures of all key sets.

In a sense the key sets are the first sets one reaches when traversing the powerset $\mathfrak{P}(M)$ levelwise.

Proposition 5. *The set $\mathscr{K}$ is an order ideal of $(\mathfrak{P}(M), \subseteq)$; i.e., $Y \in \mathscr{K}$ and $X \subseteq Y$ implies $X \in \mathscr{K}$, for all $X, Y \subseteq M$.*

Proof. Let $X \subseteq Y$ and X be a non-key set. Then exists a minimal $Z \in [X]$ with $Z \subset X$. [4] From $h(Z) = h(X)$ it follows $h(Y) = h(Y \setminus (X \setminus Z))$. Hence Y is not minimal in $[Y]$ and thus by definition not a key set. □

The definition of an order ideal is equivalent to $X \notin \mathscr{K}, X \subseteq Y \Rightarrow Y \notin \mathscr{K}$, for all $X, Y \subseteq M$. This allows to use a pruning strategy for determining the key sets. Originally the strategy we are going to apply was presented in [2], but only for a special case: as a heuristic for determining all frequent sets (which are, in our terminology, all sets with weights above a user-defined threshold). We recall this strategy, and show that it can be applied to arbitrary order ideals of the powerset of M.

Definition 9. Let $\mathscr{I}$ be an order ideal of $\mathfrak{P}(M)$. *A* candidate set for $\mathscr{I}$ it is a subset of M such that all its proper subsets are in $\mathscr{I}$.

The definition is justified by the fact that all combinations of the candidate sets can appear as $(k+1)$th level of an order ideal for which the first k levels are known. This statement is the subject of the first part of the following lemma. The second part states that non-candidate sets cannot appear at the $(k+1)$th level.

Lemma 6. *Let $\mathscr{X} \subseteq \mathfrak{P}_k(M)$, and let $\mathscr{Y}$ be the set of all candidate $(k+1)$-sets for the order ideal $\downarrow \mathscr{X} := \{Y \in \mathfrak{P}(M) | \exists X \in \mathscr{X} : Y \subseteq X\}$ (i.e., the order ideal generated by $\mathscr{X}$).*

[4] We use $X \subset Y$ to say that $X \subseteq Y$ and $X \neq Y$.

(1) *For each subset $\mathscr{Z}$ of $\mathscr{Y}$, there exists an order ideal $\mathscr{I}$ of $\mathfrak{P}(M)$ with $\mathscr{I}_k = \mathscr{X}$ and $\mathscr{I}_{k+1} = \mathscr{Z}$.*
(2) *For each order ideal $\mathscr{I}$ of $\mathfrak{P}(M)$ with $\mathscr{I}_k = \mathscr{X}$ the inclusion $\mathscr{I}_{k+1} \subseteq \mathscr{Y}$ holds.*

Proof. 1. Let $\mathscr{I} := (\downarrow \mathscr{X}) \cup \mathscr{Z}$. Let $Y \in \mathscr{I}$ and $X \subset Y$. We have to show that $X \in \mathscr{I}$. If $Y \in\downarrow \mathscr{X}$ then $X \in\downarrow \mathscr{X} \subseteq \mathscr{I}$ because $\downarrow \mathscr{X}$ is an order ideal. If $Y \in \mathscr{Z}$ then $X \in\downarrow \mathscr{X} \subseteq \mathscr{I}$ by Definition 9.

2. Suppose that there exists $Y \in \mathscr{I}_{k+1} \setminus \mathscr{Y}$. As $Y \notin \mathscr{Y}$, there exists $X \subset Y$ with $|X| = k$ and $X \notin \mathscr{I}_k$. Hence $Y \notin \mathscr{I}_{k+1}$. Contradiction. □

The efficient generation of the set of all candidate sets for the next level is described in the following proposition (→ Algorithm 2). We assume that M is linearly ordered, e.g., $M = \{1, \ldots, n\}$.

Proposition 7. *Let $\mathscr{X} \subseteq \mathfrak{P}_{k-1}(M)$. Let*

$$\widetilde{\mathscr{C}} := \{\{x_1 < x_2 < \ldots < x_k\} | \{x_1, \ldots, x_{k-2}, x_{k-1}\}, \{x_1, \ldots, x_{k-2}, x_k\} \in \mathscr{X}\};$$

and

$$\mathscr{C} := \left\{X \in \widetilde{\mathscr{C}} | \forall x \in X : X \setminus \{x\} \in \mathscr{X}\right\}.$$

Then $\mathscr{C} = \{X \in \mathfrak{P}_k(M) | X$ is candidate set for $\downarrow \mathscr{X}\}$.

Proof. The definition of C is equivalent to $C := \{x \in \widetilde{C} | X$ is candidate set for $\downarrow \mathscr{X}\}$. Hence it remains to show that all candidate sets are included in $\widetilde{C}$. Let X be a candidate set, and let $X = \{x_1, \ldots, x_k\}$ with $x_1 < \cdots < x_k$. Since X is a candidate set, all its proper subsets are in $\downarrow \mathscr{X}$, especially the two sets $\{x_1, \ldots, x_{k-2}, x_{k-1}\}$ and $\{x_1, \ldots, x_{k-2}, x_k\}$. Since they have cardinality k, they are also in $\mathscr{X}$. Hence $X \in \mathscr{I}$ by definition of $\widetilde{C}$. □

Unlike in the Apriori algorithm [2], in our application the pruning of a set cannot be determined by its properties alone, but properties of its subsets (i.e., their weights) have to be taken into account as well. This causes an additional step in the generation function (→ Algorithm 2, step 5) compared to the version presented in [2]. Based on this additional step, at each iteration the non-key sets among the candidate sets are pruned (→ Algorithm 1, step 8) by using (2) of the following proposition.

Proposition 8. *Let $X \subseteq M$.*

(1) *Let $m \in X$. Then $X \in [X \setminus \{m\}]$ if and only if $s(X) = s(X \setminus \{m\})$.*
(2) *X is a key set if and only if $s(X) \neq \min\{s(X \setminus \{m\}) | m \in X\}$.*

Proof. 1. The “if” part follows from Definition 5 (iii), the “only if” part from Definition 5 (ii).

2. From 1 we deduce that X is a key set if and only if $s(X) \neq s(X \setminus \{m\})$, for all $m \in X$. Since s is a monotonous decreasing function, this is equivalent to 2. □

A candidate set X is hence pruned when $s(X) = \min\{s(X \setminus \{m\}) | m \in X\}$ holds.

5.3. Deriving weights from already known weights

If we reach a k-set which is known not to be a key set, then we already passed along at least one of the key sets in its equivalence class in an earlier iteration. Hence we already know its weight. Using the following proposition, we determine this weight by using only weights already computed.

Proposition 9. *If X is not a key set, then*

$$s(X) = \min\{s(K) | K \in \mathscr{K}, K \subseteq X\}.$$

Proof. "$\geqslant$": Let K be a key set with $K\theta X$ and $K \subseteq X$. Then $s(X) = s(K) \geqslant \min\{s(K)|K \in \mathscr{K}, K \subseteq X\}$.

"$\leqslant$": Suppose that there exists $K \in \mathscr{K}$ with $K \subseteq X$ and $s(K) < s(X)$. Then $K \not\subseteq X$ by Definition 5, (i). Contradiction. □

Hence it is sufficient to compute the weights of the candidate sets only (by calling a function depending on the specific application → Algorithm 1, step 7). All other weights can be derived from those weights.

Now we are able to put all pieces together and to turn them into an algorithm.

6. The Titanic algorithm

The pseudo-code is given in Algorithm 1. A list of notations is provided in Table 2.

Algorithm 1. [Titanic]

```
1) Weight ({$\emptyset$});
2) $\mathscr{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) forall $m \in M$ do $\{m\}.p\_s \leftarrow \emptyset.s$;
5) $\mathscr{C} \leftarrow \{\{m\} | m \in M\}$;
6) loop begin
7)    Weigh($\mathscr{C}$);
8)    forall $X \in \mathscr{K}_{k-1}$ do $X$.closure $\leftarrow$ Closure($X$);
9)    $\mathscr{K}_k \leftarrow \{X \in \mathscr{C} | X.s \neq X.p\_s\}$;
10)   if $\mathscr{K}_k = \emptyset$ then exit loop;
11)   $k++$;
12)   $\mathscr{C} \leftarrow$ Titanic-Gen($\mathscr{K}_{k-1}$);
13) end loop;
14) return $\bigcup_{i=0}^{k-1}\{X.\mathrm{closure} | X \in \mathscr{K}_i\}$.
```

The algorithm starts with determining the weight of the empty set (step 1) and stating that it is always a key set (step 2). Then all 1-sets are candidate sets by definition (steps 4 and 5).

In later iterations, the candidate k-sets are determined by the function Titanic-Gen (step 12 ⇝ Algorithm 2) which is (except step 5) a straight-forward implementation of Proposition 7. The

Table 2
Notations used in Titanic

k	The counter which indicates the current iteration. In the kth iteration, all key k-sets are determined.
$\mathscr{K}_k$	Contains after the kth iteration all key k-sets K together with their weight $K.s$ and their closure K.closure.
$\mathscr{C}$	Stores the candidate k-sets C together with a counter $C.p\_s$ which stores the minimum of the weights of all $(k-1)$-subsets of C. The counter is used in step 9 to prune all non-key sets.

result of step 5 of Algorithm 2 will be used in step 9 of Algorithm 1 for pruning the non-key sets according to Proposition 8 (2).

Algorithm 2. (Titanic-Gen)

Input: $\mathscr{K}_{k-1}$, the set of key $(k-1)$-sets K with their weight $K.s$.
Output: $\mathscr{C}$, the set of candidate k-sets C with the values $C.p\_s := \min\{s(C \setminus \{m\}|m \in C\}$.
The variables p_s assigned to the sets $\{m_1, \ldots, m_k\}$ which are generated in step 1 are initialized by $\{m_1, \ldots, m_k\}.p\_s \leftarrow s_{\max}$.

```
1) $\mathscr{C} \leftarrow \{\{m_1 < m_2 < \cdots < m_k\}|\{m_1, \ldots, m_{k-2}, m_{k-1}\}, \{m_1, \ldots, m_{k-2}, m_k\} \in \mathscr{K}_{k-1}\}$;
2) forall $X \in \mathscr{C}$ do begin
3)     forall $(k-1)$-subsets $S$ of $X$ do begin
4)         if $S \notin \mathscr{K}_{k-1}$ then begin $\mathscr{C} \leftarrow \mathscr{C} \setminus \{X\}$; exit forall; end;
5)         $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6)     end;
7) end;
8) return $\mathscr{C}$.
```

Once the candidate k-sets are determined, the function Weigh($\mathscr{X}$) is called to compute, for each $X \in \mathscr{X}$, the weight of X and stores it in the variable $X.s$ (step 7).

Remark. In the case of concept lattices, Weigh determines the weights (i.e., the supports) of all $X \in \mathscr{X}$ with *a single pass* of the context (see Section 7.1). This is the reason why we call the function Weigh for a set of sets instead of calling it for each set separately. In general, computing the weights of different sets simultaneously may or may not be more efficient than doing it separately, depending on the application.

Algorithm 3. (Closure (X) for $X \in \mathscr{K}_{k-1}$)

```
1) $Y \leftarrow X$;
2) forall $m \in X$ do $Y \leftarrow Y \cup (X \setminus \{m\})$.closure;
3) forall $m \in M \setminus Y$ do begin
4)     if $X \cup \{m\} \in \mathscr{C}$ then $s \leftarrow (X \cup \{m\}).s$
5)         else $s \leftarrow \min\{K.s|K \in \mathscr{K}, K \subseteq X \cup \{m\}\}$;
6)     if $s = X.s$ then $Y \leftarrow Y \cup \{m\}$
7) end;
8) return $Y$.
```

For those sets which remained from the pruning (step 9) in the previous pass (and which are now known to be key sets), their closures are computed (step 8 $\rightsquigarrow$ Algorithm 3). The Closure function (Algorithm 3) is a straight-forward implementation of Proposition 4 (steps 3–7) and Proposition 9 (step 5) plus an additional optimization (step 2).

In step 9 of Algorithm 1, all candidate k-sets which are not key sets are pruned according to Proposition 8 (2). Algorithm 1 terminates, if there are no key k-sets left (step 10). Otherwise the next iteration begins (step 11).

The correctness of the algorithm is proved by the theorems in the previous section. Examples for the algorithm are given in Section 7.

7. Computing (Iceberg) concept lattices with Titanic

In the sequel we will show that, for a given formal context, the support function fulfills the conditions of Definition 5 for being compatible to the closure operator $h(X) := X''$. Hence computing concept lattices is a typical application of the problem. We will also discuss how to modify the closure operator such that the problem description applies to iceberg concept lattices as well.

We demonstrate the Titanic algorithm by two examples: computing a concept lattice, and computing an iceberg concept lattice. For other applications (for instance those listed in Section 8), only the Weigh function has to be adapted.

7.1. Computation of concept lattices

In the following, we will use the composed function $B \mapsto B''$, for $B \subseteq M$. It is (by Theorem 3) a closure operator on M. The related closure system (i.e., the set of all $B \subseteq M$ with $B'' = B$) is by Lemma 2 exactly the set of the intents of all concepts of the context. The structure of the concept lattice is hence already determined by this closure system. Therefore we restrict ourselves to the computation of the closure system of all concept intents in the sequel. The computation makes extensive use of the support function introduced in Definition 3. We show that the support function fulfills the conditions of Definition 5.

Lemma 10. *Let* $X, Y \subseteq M$.

(1) $X \subseteq Y \Rightarrow \mathrm{supp}(X) \geqslant \mathrm{supp}(Y)$.
(2) $X'' = Y'' \Rightarrow \mathrm{supp}(X) = \mathrm{supp}(Y)$.
(3) $X \subseteq Y \wedge \mathrm{supp}(X) = \mathrm{supp}(Y) \Rightarrow X'' = Y''$.

Proof. 1. Let $X \subseteq Y$. Then $Y' \subseteq X'$ by Lemma 1, which implies

$$\mathrm{supp}(Y) = \frac{|Y'|}{|G|} \leqslant \frac{|X'|}{|G|} = \mathrm{supp}(X).$$

2. $X\theta Y \iff X'' = Y'' \iff X''' = Y''' \iff X' = Y' \Rightarrow s(X) = \frac{|X'|}{|G|} = \frac{|Y'|}{|G|} = s(Y)$.

Algorithm 4. (The Weigh algorithm for concept lattics)

```
1) forall $X \in \mathscr{X}$ do $X.s \leftarrow 0$;
2) forall $g \in G$ do
3)     forall $X \in \text{subsets}(g', \mathscr{X})$ do $X.s++$;
4) forall $X \in \mathscr{X}$ do $X.s \leftarrow \frac{X.s}{|G|}$;
```

3. $\text{supp}(X) = \text{supp}(Y)$ implies $|X'| = |Y'|$, and $X \subseteq Y$ implies $X' \supseteq Y'$. Hence $X' = Y'$, since X' and Y' are finite. It follows $X'' = Y''$. □

Corollary 11. *The support count is a weight function which is compatible with the closure operator* $X \mapsto X''$.

Thus we can use Titanic for computing concept lattices. In this special application, we can benefit from two optimizations:

(1) In Algorithm 1, we can—in the case of (iceberg) concept lattices—replace step 1 by
 1′) $\emptyset.s \leftarrow 1$
since we know that $\text{supp}(\emptyset) = 1$. We avoid one call of the Weigh function.

(2) For concept lattices, Weigh determines the weights, that is, the supports, of all $X \in \mathscr{X}$ with a single pass over the context. This is (together with the fact that only $\max\{|X| \mid X \subseteq M \text{ is candidate set}\}$ passes are needed) the reason for the efficiency of Titanic. The Weigh algorithm for concept lattices is given in Algorithm 4. Subsets$(Y, \mathscr{X})$ returns, for $Y \subseteq M$ and $\mathscr{X} \subseteq \mathfrak{P}(M)$, all $X \in \mathscr{X}$ with $Y \subseteq X$. It uses a tree structure with hash tables (as described in [30]) to efficiently encode $\mathscr{X}$.

	edible (e)	poisonous (p)	cap shape: convex (c)	cap shape: flat (l)	cap surface: fibrous (i)
Mushroom 1	☒		☒		
Mushroom 2	☒		☒		☒
Mushroom 3	☒			☒	☒
Mushroom 4	☒			☒	☒
Mushroom 5	☒		☒		☒
Mushroom 6	☒		☒		
Mushroom 7		☒		☒	☒
Mushroom 8		☒		☒	☒
Mushroom 9		☒		☒	☒
Mushroom 10		☒		☒	

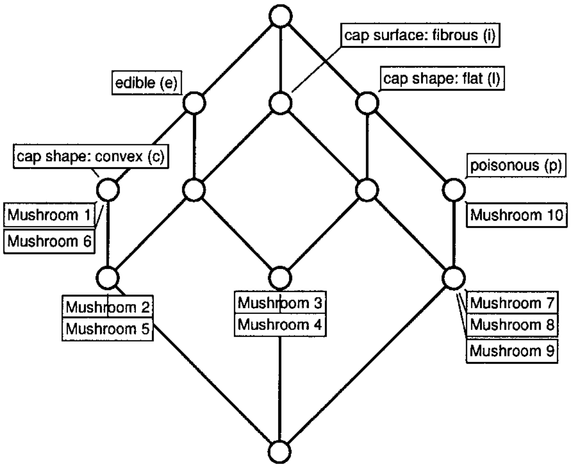

Fig. 8. Example for the Titanic algorithm.

Example. For explaining how Titanic works, we will use the mushroom example in Fig. 1 again, but will reduce it further to the first five attributes (see Fig. 8).

In the first pass, the algorithm deals with the empty set and all 1-sets. It returns the results for $k = 0$ and $k = 1$:

$k = 0$:

Step 1		Step 2
X	$X.s$	$X \in \mathcal{K}_k$?
$\emptyset$	1	Yes

$k = 1$:

Steps 4 + 5		Step 7	Step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e\}$	1	6/10	Yes
$\{p\}$	1	4/10	Yes
$\{c\}$	1	4/10	Yes
$\{l\}$	1	6/10	Yes
$\{i\}$	1	7/10	Yes

Step 8 returns: $\emptyset$.closure $\leftarrow$ $\emptyset$.

Then the algorithm repeats the loop for $k = 2$, 3, and 4:

$k = 2$:

Step 12		Step 7	Step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e, p\}$	4/10	0	Yes
$\{e, c\}$	4/10	4/10	No
$\{e, l\}$	6/10	2/10	Yes
$\{e, i\}$	6/10	4/10	Yes
$\{p, c\}$	4/10	0	Yes
$\{p, l\}$	4/10	4/10	No
$\{p, i\}$	4/10	3/10	Yes
$\{c, l\}$	4/10	0	Yes
$\{c, i\}$	4/10	2/10	Yes
$\{l, i\}$	6/10	5/10	Yes

Step 8 returns: $\{e\}$.closure $\leftarrow \{e\}$, $\{p\}$.closure $\leftarrow \{p, l\}$, $\{c\}$.closure $\leftarrow \{c, e\}$, $\{l\}$.closure $\leftarrow \{l\}$ $\{i\}$.closure $\leftarrow \{i\}$.

$k = 3$:

Step 12		Step 7	Step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e, l, i\}$	2/10	2/10	No
$\{p, c, i\}$	4/10	0	Yes
$\{c, l, i\}$	4/10	0	Yes

Step 8 returns: $\{e,p\}$.closure $\leftarrow \{e,p,c,l,i\}$, $\{e,l\}$.closure $\leftarrow \{e,l,i\}$, $\{e,i\}$.closure $\leftarrow \{e,i\}$, $\{p,c\}$.closure $\leftarrow \{e,p,c,l,i\}$, $\{p,i\}$.closure $\leftarrow \{p,l,i\}$, $\{c,l\}$.closure $\leftarrow \{e,p,c,l,i\}$, $\{c,i\}$.closure $\leftarrow \{e,c,i\}$, $\{l,i\}$.closure $\leftarrow \{l,i\}$.

$k = 4$:

Step 12 returns the empty set. Hence there is nothing to Weigh in step 7. Step 9 sets $\mathcal{K}_4$ equal to the empty set; and in step 10, the loop is exited.

Step 8 returns: $\{p,c,i\}$.closure $\leftarrow \{e,p,c,l,i\}$, $\{c,l,i\}$.closure $\leftarrow \{e,p,c,l,i\}$.

Finally the algorithm collects all concept intents (step 14):

$$\emptyset, \{e\}, \{p,l\}, \{c,e\}, \{l\}, \{i\}, \{e,p,c,l,i\}, \{e,l,i\}, \{e,i\}, \{p,l,i\}, \{e,c,i\}, \{l,i\}$$

(which are exactly the intents of the concepts of the concept lattice in Fig. 8). The algorithm determined the support of $5 + 10 + 3 = 18$ attribute sets in three passes of the database.

7.2. Equipping Titanic for iceberg concept lattices

The structure of an iceberg concept lattice is determined by the semi-lattice of its frequent intents. If we add the set M (which is not frequent in general) to the set of frequent intents, it becomes a closure system. Lemma 12 presents its closure operator.

Lemma 12. *Let* $\mathbb{K} := (G, M, I)$ *be a context, and let* min supp $\in [0, 1]$. *The set* $\mathcal{F} := \{B \subseteq M | (A, B) \in \mathfrak{B}(\mathbb{K}), \text{supp}(B) \geqslant \text{min supp}\} \cup \{M\}$ *is a closure system on M. Its closure operator is given by* $h(X) := X''$ *if* $\text{supp}(X) \geqslant$ min supp *and* $h(X) := M$ *else. The weight function* $s(X) := \text{supp}(X)$ *if* $\text{supp}(X) \geqslant$ min supp *and* $s(X) := -1$ *else is compatible with the closure operator.*

Proof. $\tilde{\mathcal{F}} := \{B \subseteq M | \text{supp}(B) \geqslant \text{min supp}\} \cup \{M\}$ is a closure system, since it is closed under arbitrary intersections. $\text{Int}(\mathbb{K}) := \{B \subseteq M | (A, B) \in \mathfrak{B}(\mathbb{K})\}$ is a closure system by Theorem 3. Hence $\mathcal{F}$ is—as intersection of the two closure systems $\widetilde{\mathcal{F}}$ and $\text{Int}(\mathbb{K})$—also a closure system. Verifying that h is the related closure operator and that s is compatible is straightforward. □

The lemma shows that the Titanic algorithm as presented in Section 6 can directly be applied to iceberg concept lattices. However we can benefit from the fact that weight -1 indicates that the closure of the set is the whole set M. In this case we can improve the algorithm. The improved version is discussed now.

Algorithm 5. (Titanic improved for iceberg concept lattices)

```
1) $\emptyset.s \leftarrow 1$;
2) $\mathscr{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) forall $m \in M$ do $\{m\}.p\_s \leftarrow \emptyset.s$;
5) $\mathscr{C} \leftarrow \{\{m\} | m \in M\}$;
6) loop begin
7)    Weigh $(\mathscr{C})$;
8)    forall $X \in \mathscr{K}_{k-1}$ do $X.\mathrm{closure} \leftarrow \mathrm{closure}(X)$;
9)    $\mathscr{K}_k \leftarrow \{X \in \mathscr{C} | X.s \neq X.p\_s\}$;
10)    if $\{X \in \mathscr{K}_k | X.s \neq -1\} = \emptyset$ then exit loop;
11)    $k++$;
12)    $\mathscr{C} \leftarrow$ Titanic-Gen$(\mathscr{K}_{k-1})$;
13) end loop;
14) return $\bigcup_{i=0}^{k-1}\{X.\mathrm{closure} | X \in \mathscr{K}_i, X.s \neq -1\}$.
```

Algorithm 6. (Titanic-Gen improved for iceberg concept lattices)

Input: $\mathscr{K}_{k-1}$, the set of key $(k-1)$-sets K with their support $K.s$.
Output: $\mathscr{C}$, the set of candidate k-sets C with the values $C.p\_s := \min\{s(C \setminus \{m\} | m \in C\}$.
The variables p_s assigned to the sets $\{m_1, \ldots, m_k\}$ which are generated in step 1 are initialized by $\{m_1, \ldots, m_k\}.p\_s \leftarrow 1$.

```
1) $\mathscr{C} \leftarrow \{\{m_1 < m_2 < \ldots < m_{k-1} < m_k\} | \{m_1, \ldots, m_{k-2}, m_{k-1}\}, \{m_1, \ldots,$
   $m_{k-2}, m_k\} \in \{K \in \mathscr{K}_{k-1} | K.s \neq -1\}\}$;
2) forall $X \in \mathscr{C}$ do begin
3)    forall $(k-1)$-subsets $S$ of $X$ do begin
4)       if $S \notin \mathscr{K}_{k-1}$ or $S.s = -1$ then begin $\mathscr{C} \leftarrow \mathscr{C} \setminus \{X\}$; exit forall; end;
5)       $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6)    end;
7) end;
8) return $\mathscr{C}$.
```

Algorithm 7. (closure for iceberg conept lattices)

```
1) if $X.s = -1$ then return $M$;
2) $Y \leftarrow X$;
3) forall $m \in X$ do $Y \leftarrow Y \cup (X \setminus \{m\}).\mathrm{closure}$;
4) forall $m \in M \setminus Y$ do begin
5)    if $X \cup \{m\} \in \mathscr{C}$ then $s \leftarrow (X \cup \{m\}).s$
6)          else $s \leftarrow \min\{K.s | K \in \mathscr{K}, K \subseteq X \cup \{m\}\}$;
7)    if $s = X.s$ then $Y \leftarrow Y \cup \{m\}$
8) end;
9) return $Y$.
```

Algorithm 5 differs from Algorithm 1 in steps 1, 10, and 14; Algorithm 6 differs from Algorithm 2 in steps 1 and 4; and Algorithm 7 is extending Algorithm 3 by step 1. We discuss these differences step by step:

- Algorithm 5, step 1: See the remark about the first optimization in Section 7.1.
- Algorithm 5, step 10: The loop can be exited when no *or only infrequent* key sets remain, as they are not used for generating candidate sets in the next iteration (see Algorithm 6, step 1).
- Algorithm 5, step 14: The algorithm returns only frequent intents, i.e. only closures of frequent key sets.
- Algorithm 6, step 1: Only frequent key sets are used to construct new candidate sets. See next item.
- Algorithm 6, step 4: S is a candidate set only if all $(k-1)$-subsets of S are frequent key sets, because sets containing an infrequent key set are known not to be key sets.
- Algorithm 7, step 1: If the weight of a set is -1, its closure must be M by Lemma 12.

As before, the function weigh($\mathscr{X}$) determines, in one pass of the context, for each $X \in \mathscr{X}$ the support of X and stores it in the variable $X.s$. If $s(X) <$ minsupp, then WEIGH returns $X.s \leftarrow -1$.

Example. Although TITANIC only needs three passes of the database to compute the iceberg lattice in Fig. 3 (and four passes for the one in Fig. 5), we decided not to use it as example for explaining the mechanism of TITANIC for iceberg lattices. The reason is, that at the first pass the algorithm has to handle 80 candidate itemsets of size one. Of course, this is no problem in praxis, but is too large for demonstration purposes. Therefore we reuse the context in Fig. 8, and show the computation of its iceberg concept lattice for minsupp = 30%.

In the first pass, the algorithm deals with the empty set and all 1-sets. It returns the results for $k=0$ and $k=1$. As no infrequent sets are considered here, the results are exactly the same as in Example 7.1:

$k=0$:

Step 1		Step 2
X	$X.s$	$X \in \mathscr{K}_k$?
$\emptyset$	1	Yes

$k=1$:

Steps 4 + 5		Step 7	Step 9
X	$X.p\_s$	$X.s$	$X \in \mathscr{K}_k$?
$\{e\}$	1	6/10	Yes
$\{p\}$	1	4/10	Yes
$\{c\}$	1	4/10	Yes
$\{l\}$	1	6/10	Yes
$\{i\}$	1	7/10	Yes

Step 8 returns: $\emptyset$.closure $\leftarrow \emptyset$.

Then the algorithm repeats the loop for $k = 2$. Here, the first infrequent sets are reached:

$k = 2$:

Step 12		Step 7	Step 9
X	$X.p\_s$	$X.s$	$X \in \mathscr{K}_k$?
$\{e,p\}$	4/10	-1	Yes
$\{e,c\}$	4/10	4/10	No
$\{e,l\}$	6/10	-1	Yes
$\{e,i\}$	6/10	4/10	Yes
$\{p,c\}$	4/10	-1	Yes
$\{p,l\}$	4/10	4/10	No
$\{p,i\}$	4/10	3/10	Yes
$\{c,l\}$	4/10	-1	Yes
$\{c,i\}$	4/10	-1	Yes
$\{l,i\}$	6/10	5/10	Yes

Step 8 returns: $\{e\}$.closure $\leftarrow \{e\}$, $\{p\}$.closure $\leftarrow \{p,l\}$, $\{c\}$.closure $\leftarrow \{c,e\}$, $\{l\}$.closure $\leftarrow \{l\}$, $\{i\}$.closure $\leftarrow \{i\}$.

Remark. As the weight of the key sets $\{e,p\}$, $\{e,l\}$, $\{c,l\}$, and $\{c,i\}$ is -1, we know that these sets are infrequent (with respect to our minimum support threshold of 30%). In the corresponding closure system, they will hence generate the whole set M. These infrequent key sets are important if we want to provide a basis for association rules. See [41] for details. If our aim is conceptual clustering, we can neglect these infrequent key sets and can improve the performance of the algorithm by modifying step 9 in Algorithm 5 to

9a) $\mathscr{K}_k \leftarrow \{X \in \mathscr{C} | X.s \neq X.p\_s$ and $X.s \neq 1\}$.

This would yield 'yes' instead of 'no' in the last column for the five sets mentioned above.

$k = 3$:

Step 12 returns the empty set (because of the condition $K.s \neq -1$ in step 1 of Algorithm 2). Hence there is nothing to Weigh in step 7. Step 9 sets $\mathscr{K}_3$ equal to the empty set; and in step 10, the loop is exited.

Step 8 returns: $\{e,p\}$.closure $\leftarrow M$, $\{e,l\}$.closure $\leftarrow M$, $\{e,i\}$.closure $\leftarrow \{e,i\}$, $\{p,c\}$.closure $\leftarrow M$, $\{p,i\}$.closure $\leftarrow \{p,l,i\}$, $\{c,l\}$.closure $\leftarrow M$, $\{c,i\}$.closure $\leftarrow M$, $\{l,i\}$.closure $\leftarrow \{l,i\}$.

Finally the algorithm collects all frequent concept intents (step 14):

$$\emptyset, \{e\}, \{p,l\}, \{c,e\}, \{l\}, \{i\}, \{e,i\}, \{p,l,i\}, \{l,i\}$$

The resulting concept iceberg lattice is shown in Fig. 9.

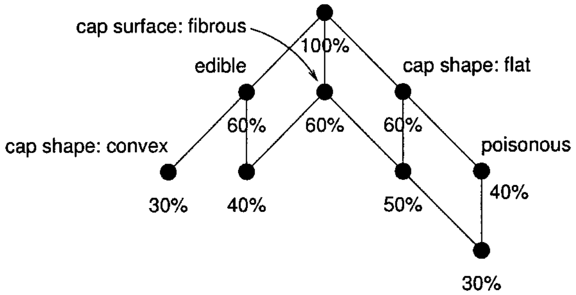

Fig. 9. Iceberg concept lattice for the context in Fig. 8 for minsupp = 30%.

8. Some typical applications

In Section 3, we have already discussed the use of (iceberg) concept lattices for knowledge discovery and conceptual clustering. Here we present two examples, in which iceberg concept lattices have been applied.

8.1. Database marketing

The purpose of database marketing is the study of customers and their buying behavior in order to create and validate marketing strategies. In [16], the use of iceberg concept lattices for database marketing in a Swiss department store is discussed in more detail. In that scenario, the object set *G* consists of all customers of the warehouse paying by credit card, and the attribute set *M* consists of attributes describing the customers (e.g., 'lives in Western Switzerland') and their buying behavior (e.g., 'has spent more than 1000 Swiss francs in the last year'). For a given set *X* of attributes, the weight function returns the number of customers fulfilling all attributes in *X*. By decreasing the minimum support, one can study the customer behavior in more and more detail. In Fig. 10, for instance, the customers of the warehouse are clustered according to their year of birth. The minimum support threshold is set to 0.3, i.e., all concepts whose extents do not comprise at least 30% of all customers, are pruned.

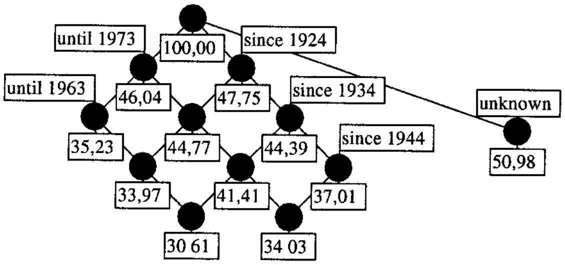

Fig. 10. Iceberg concept lattice for customers clustered by their year of birth.

8.2. Ontology learning

Ontologies are "explicit specification(s) of a conceptualization" [15]. They usually consist of a set of concepts (not to be confused with formal concepts from FCA), a hierarchical is-a relation and other (non-hierarchical) relations between the concepts, and eventually axioms describing constraints on the relations and concepts. One task in learning ontologies from data is the construction of the is-a hierarchy. Suppose that the concepts are already learned (e.g., by applying linguistic and statistical methods [24]) and stored in the set M. The set G contains instances, or documents annotated with the concepts. The relation I indicates if an instance belongs to a concept, or if a document is annotated with a concept. In [38], this approach has been used in FCA–Merge, a technique for supporting the merging of ontologies. There, Titanic uses the weight function which assigns to a set X of ontology concepts the number of documents/instances related to all concepts in X. The resulting iceberg concept lattice provides an is-a hierarchy on the set of the ontology concepts. Additionally, it suggests new concepts which may simplify the structure of the concept hierarchy.

The use of (iceberg) concept lattices is not only restricted to knowledge discovery. Here we give some more examples of typical applications, in which FCA has been successfully applied in the past (before the introduction of Titanic). Their purpose is to show that the weight function (whose existence is a necessary condition for the applicability of Titanic) naturally appears in a wide variety of domains.

8.3. Configuration space analysis

In software re-engineering, one task is to analyze the source code of a given program where no (or relatively few) documentation is given. In [19], the use of FCA for analyzing the configuration space of C++ programs is discussed. In the described scenario, iceberg concept lattices could be introduced quite naturally. The set G of objects contains the lines of code, the set M consists basically of the C++ preprocessor symbols which appear in the code, and the relation I indicates which lines of code are governed by which preprocessor symbols. Instead of computing the whole concept lattice, one can restrict the computation to the top-level groupings of code pieces by using Titanic. The weight function returns, for a set X of preprocessor symbols, the number of lines of code which are governed by all preprocessor symbols in X.

8.4. Transformation of class hierarchies

In object-oriented languages, one aim is to simplify the class hierarchy according to a (number of) given program(s). In [36], this problem has been attacked by using concept lattices. In the scenario, the set M of attributes contains all data members and methods of a given class hierarchy, and the set G of objects consists of all variables and pointers of the program(s). The relation I basically indicates which variables and pointers are related to which data members and methods.The resulting concept lattice provides an improved hierarchy which can be used for restructuring the class hierarchy according to software engineering principles without the need to modify the source code. The computation of the concept lattice can be done by using as weight function the function which returns, for a given set X of data members and methods, the number of variables and pointers related to all elements in X.

Another situation where a weight function arises naturally in the computation of a closure system is the following. This scenario is more difficult to state in terms of a formal context.

8.5. Discovery of functional dependencies

One important task of logical database tuning is the discovery of minimal functional dependencies from database relations [17,21]. This is equivalent to computing a closure system on the set M of all database attributes. The closed sets are just those which are closed under all functional dependencies which hold in the database. Titanic can be applied for this computation, using as weight of a given attribute set X the minimal number of rows which have to be deleted from the database such that X is closed under all functional dependencies which are valid for the remaining rows. This weight function is derived from the g_3 measure introduced in [18]. For this application, all 'min' in this paper have to be replaced by 'max' (refer to Remark 2).

9. Complexity and experimental evaluation

There are several algorithms known for computing concept lattices [11,14,28,29,31–33]. The most efficient algorithm for practical applications at the best of our knowledge is Ganter's Next-Closure algorithm [11]; the algorithm with the best worst-case complexity is the one from Nourine and Raynaud presented in [29]. The latter one substantiates in an efficient way an approach proposed by Wille [46]. In this section, we will compare Titanic with these two algorithms.

The problem of computing concept lattices has exponential worst-case complexity: The context $\mathbb{K} := (\{1, \ldots, n\}, \{1, \ldots, n\}, \neq)$ has n objects and n attributes, while its concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$ has 2^n concepts. Therefore all three algorithms have an exponential complexity. However, for practical purposes, it is interesting to examine the situation in more detail. In the sequel, we assume that $|M| \leqslant |G|$.

Ganter's algorithm computes the concepts sequentially. In [11] is shown that the complexity for computing one concept is in $\mathrm{O}(|G| \cdot |M|^2)$, so that the overall complexity could be stated as $\mathrm{O}(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (|G| \cdot |M|^2))$. For each concept, the context has to be accessed. If we consider additionally the access time db of the formal context (which can be significantly large when the context is too large to be stored in main memory!), we obtain $O(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (db + |G| \cdot |M|^2))$.

The algorithm of Nourine and Raynaud also computes the concepts sequentially. For each concept, the algorithm needs time $O((|M| + |G|) \cdot |G|)$, thus improving Ganter's worst-case complexity. Both algorithms need to access the context for each concept to be computed: If we add the access time db of the formal context to Nourine and Raynaud's algorithm, it is in $O(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (db + (|M| + |G|) \cdot |G|))$. On the other hand, Nourine and Raynaud's algorithm needs exponential space, since the whole lattice must be stored during run-time; while Next-Closure needs the context only, and has thus linear space complexity.

Both algorithms have different benefits. While Next-Closure needs only linear space, Nourine and Raynaud's algorithm provides the best worst-case complexity known so far. On the other hand, Next-Closure can be easily adapted to efficiently compute iceberg lattices, while the structure of Nourine and Raynaud's algorithm prohibits this. Furthermore, for the latter algorithm, the need to access the results computed so far makes it impractical for very large databases (contexts). Therefore, we will compare TITANIC in the experimental evaluation with Ganter's Next-Closure algorithm only.

From a complexity point of view, TITANIC is in between those two algorithms. Its worst-case space complexity is reached, when all $\lfloor\frac{|M|}{2}\rfloor$-sets are candidate sets. Then all these

$$\binom{|M|}{\lfloor\frac{|M|}{2}\rfloor} = \frac{|M|\cdot\ldots\cdot\left(\lfloor\frac{|M|}{2}\rfloor+1\right)}{\lceil\frac{|M|}{2}\rceil\cdot\ldots\cdot 1}$$

sets have to be stored. This is the widest level of the powerset of $|M|$, and its width grows exponentially relatively to $|M|$.

TITANIC's time complexity can be determined as follows: The algorithm accesses the context as often as the size L of the largest candidate set is. This size is bounded by $|M|$, the height of the powerset of M. At each access, the algorithm considers a number of candidate sets. Let N be the maximal number of candidate sets considered at one of the accesses of the context. Then the time complexity is $O(L\cdot(db+N\cdot|G|\cdot|M|))$. By using the upper limits for L and n, we obtain

$$O\left(|M|\cdot\left(db+\binom{|M|}{\lfloor\frac{|M|}{2}\rfloor}\cdot|G|\cdot|M|\right)\right).$$

We see that the number of accesses of the context is at most $|M|$ (rather than $2^{|M|}$ as for the other two algorithms), which is especially important, when the context is so large that it doesn't fit into main memory. In that case, db can be a significant (or even the dominant) time factor.

The results show TITANIC's worst-case complexity. In praxis the values for L and n are usually much lower. Especially for n (which contributes the exponentiality), the upper limit is, in the average case for computing iceberg concept lattices, the number of 2-itemsets, which is at most

$$\binom{|M|}{2} = \frac{|M|\cdot(|M|-1)}{2}.$$

We evaluated TITANIC experimentally also. For our evaluation, a version of the TITANIC algorithm was implemented in C++ together with a rewriting of Lindig's C version of Next-Closure [20] The comparisons took place on a Pentium III running at 600 MHz, with 512 MB of main memory, and were performed on the MUSHROOM (8416 objects, 80 attributes) and INTERNET (10,000 objects, 141 attributes) databases, both available from the UCI KDD Archive (http://kdd.ics.uci.edu/), with a varying number of objects.

The results are visualized in Figs. 11 and 12, and listed in detail in Table 3. They show that on the weakly correlated INTERNET database, Next-Closure is faster for few attributes, but takes three times the time of TITANIC for the whole dataset. On the strongly correlated MUSHROOMS database, TITANIC is two to five times faster than Next-Closure.

In [5], we showed that a modified version of TITANIC for computing association rules called PASCAL (which computes all frequent itemsets, and not only the closed ones) outperforms the

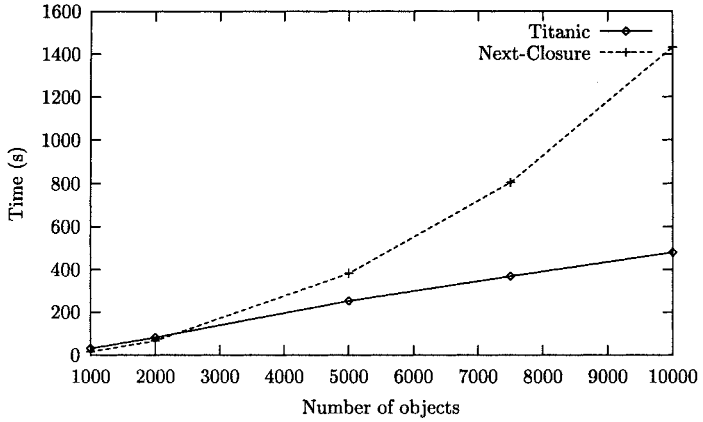

Fig. 11. Comparison of Titanic and Next-Closure on the Internet database.

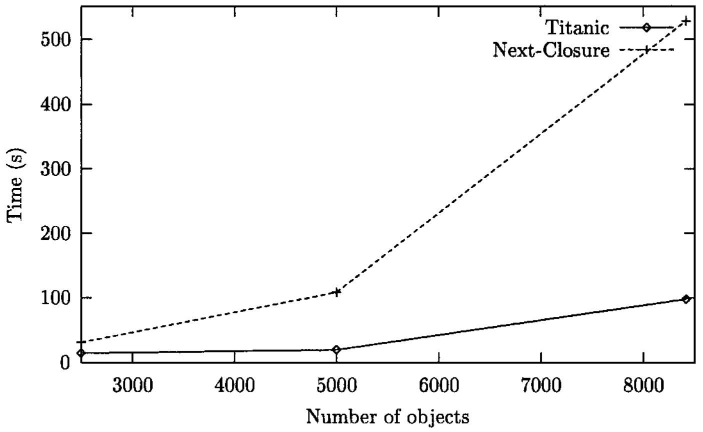

Fig. 12. Comparison of Titanic and Next-Closure on the Mushrooms database.

Table 3
Database characteristics and evaluation results

Database	# of objects	# of attraction	# of concepts	Computation time (s)	
				Next-closure	Titanic
Internet	1000	141	15,107	16.49	31.29
	2000	141	31,719	66.32	82.70
	5000	141	73,026	381.95	253.00
	7500	141	100,706	803.17	368.44
	10,000	141	124,574	1431.86	480.34
Mushrooms	2500	79	5,394	31.13	14.87
	5000	79	9064	108.38	20.14
	8416	80	32,086	527.74	97.93

algorithms Apriori [2] and Max-Miner [6] on strongly correlated data sets (and is comparable with those algorithms on weakly correlated data sets).

The problem of computing concept lattices has exponential complexity. This shows that one cannot expect from any algorithm, however robust it is claimed to be, that it solves the problem in reasonable time in the worst case. However our experimental results with Titanic show that under normal conditions (and if handled with care) a strong and waterproof algorithm may improve the exploration of unknown regions of knowledge.

10. Conclusion

The paper provides two contributions: iceberg concept lattices and the Titanic algorithm. It shows the use of iceberg concept lattices as a conceptual clustering method, a condensed representation of frequent itemsets, and an efficient visualization technique for conceptual hierarchies derived from very large databases. Titanic is presented as an algorithm for efficiently determining closure systems for a given weight function. Typical examples for its application are listed in the paper. The mathematical foundations of the algorithm are introduced, and the algorithm is experimentally evaluated.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of SIGMOD Conference, 1993, pp. 207–216.

[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of VLDB Conference, 1994, pp. 478–499 (Expanded version in IBM Report RJ9839).

[3] A. Arnauld, P. Nicole, La logique ou l'art de penser—contenant, outre les règles communes plusieurs, observations nouvelles, propresà former le jugement, Ch Saveux, Paris, 1668.

[4] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, L. Lakhal, Mining minimal non-redundant association rules using frequent closed itemsets, in: Proceedings of the 1st International Conference on CL (6th International Conference on Database Systems), in: J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagiv, P.J. Stuckey (Eds.), Computational Logic—CL, LNAI 1861, Springer, Heidelberg, 2000, pp. 972–986.

[5] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal, Mining frequent patterns with counting inference, SIGKDD Explorations, Special Issue on Scalable Algorithms 2 (2) (2000) 71–80.

[6] R.J. Bayardo, Efficiently mining long patterns from databases, in: Proceedings of SIGMOD'98, 1998, pp. 85–93.

[7] K. Becker, G. Stumme, R. Wille, U. Wille, M. Zickwolff, Conceptual information systems discussed through an IT-security tool, in: Proceedings of EKAW'00, in: R. Dieng, in: O. Corby (Eds.), Knowledge Engineering and Knowledge Management: Methods Models and Tools, LNAI 1937, Springer, Heidelberg, 2000, pp. 352–365.

[8] C. Carpineto, G. Romano, GALOIS: an order-theoretic approach to conceptual clustering, in: Proceedings ICML, Machine Learning, Morgan Kaufmann Prublishers, 1993, pp. 33–40.

[9] R. Cole, G. Stumme, CEM: a conceptual e-mail manager, in: Proceedings ICCS'00, in: B. Ganter, in: G.W. Mineau (Eds.), Conceptual Structures: Logical, Linguistic, and Computational Issues, LNAI 1867, Springer, Heidelberg, 2000, pp. 438–452.

[10] H. Dicky, C. Dony, M. Huchard, T. Libourel, On automatic class insertion with overloading, OOPSLA (1996) 251–267.

[11] B. Ganter, K. Reuter, Finding all closed sets: A general approach, in: Order, Kluwer Academic Publishers, Amsterdam, 1991, pp. 283–290.

[12] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, Heidelberg, 1999.

[13] R. Godin, H. Mili, G. Mineau, R. Missaoui, A. Arfi, T. Chau, Design of class hierarchies based on concept Galois lattices, TAPOS 4 (2) (1998) 117–134.

[14] R. Godin, R. Missaoui, An incremental concept formation approach for learning from databases, TCS 133 (2) (1994) 387–419.

[15] T. Gruber, Towards principles for the design of ontologies used for knowledge sharing, International Journal of Human and Computer Studies 46 (2/3) (1997) 293–310.

[16] J. Hereth, G. Stumme, U. Wille, R. Wille, Conceptual knowledge discovery and data analysis, in: Proceedings ICCS2000, in: B. Ganter, in: G. Mineau (Eds.), Conceptual Structures: Logical, Linguistic, and Computational Structures, LNAI 1867, Springer, Heidelberg, 2000, pp. 421–437.

[17] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, TANE: an efficient algorithm for discovering functional and approximate dependencies, The Computer Journal 42 (2) (1999) 100–111.

[18] J. Kivinen, H. Mannila, Approximate inference of functional dependencies from relations, TCS 149 (1) (1995) 129–149.

[19] M. Krone, G. Snelting, On the inference of configuration structures from source code, in: Proceedings 16th International Conference on Software Engineering, IEEE Computer Society Press, May 1994, pp. 49–57.

[20] Ch. Lindig, Concepts, ftp://ftp.ips.cs.tu-bs.de/pub/local/softech/misc/concepts-0.3d.tar.gz, 1997 (Open Source implementation of concept analysis in C).

[21] S. Lopes, J.-M. Petit, L. Lakhal, Efficient discovery of functional dependencies and Amstrong relations, in: Proceedings EDBT 2000, LNCS 1777, Springer, Heidelberg, 2000, pp. 350–364.

[22] M. Luxenburger, Implications partielles dans un contexte, Mathématiques, Informatique et Sciences Humaines 29 (113) (1991) 35–55.

[23] K. Mackensen, U. Wille, Qualitative text analysis supported by conceptual data systems, Quality and Quantity: International Journal of Methodology 2 (33) (1999) 135–156.

[24] A. Mädche, S. Staab, Mining ontologies from text, in: Proceedings EKAW'00, in: R. Dieng, in: O. Corby (Eds.), Knowledge Engineering and Knowledge Management: Methods, Models, and Tools, LNAI 1937, Springer, Heidelberg, 2000, pp. 189–202.

[25] H. Mannila, H. Toivonen, Levelwise search and borders of theories in knowledge discovery, Data Mining and Knowledge Discovery 1 (3) (1997) 241–258.

[26] R.S. Michalski, Knowledge acquisition through conceptual clustering: a theoretical framework and an algorithm for partitioning data into conjunctive concepts, Policy Analysis and Information Systems 4 (3) (1980) 219–244.

[27] G. Mineau, R. Godin, Automatic structuring of knowledge bases by conceptual clustering, IEEE Transactions on Knowledge and Data Engineering 7 (5) (1995) 824–829.

[28] M. Missikoff, M. Scholl, An algorithm for insertion into a lattice: application to type classification, in: Proceedings rd International Conference FODO 1989, LNCS 367, Springer, Heidelberg, 1989, pp. 64–82.

[29] L. Nourine, O. Raynaud, A fast algorithm for building lattices, Information Processing Letters 71 (1999) 199–204.

[30] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Pruning closed itemset lattices for association rules, 14èmes Journes Bases de Donnes Avances (BDA'98), Hammamet, Tunisia, 26–30 October, 1998.

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

[32] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules, in: Proceedings ICDT'98, LNCS 1540, Springer, Heidelberg, 1999, pp. 398–416.

[33] J. Pei, J. Han, R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets, in: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 21–30.

[34] P. Scheich, M. Skorsky, F. Vogt, C. Wachter, R. Wille, Conceptual data systems, in: O. Opitz, B. Lausen, R. Klar (Eds.), Information and Classification, Springer, Berlin, Heidelberg, 1993, pp. 72–84.

[35] I. Schmitt, G. Saake, Merging inheritance hierarchies for database integration, in: Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, 20–22 August, 1998, pp. 122–131.

[36] G. Snelting, F. Tip, Reengineering class hierarchies using concept analysis, in: Proceedings ACM SIGSOFT Symposium on the Foundations of Software Engineering, November 1998, pp. 99–110.

[37] S. Strahringer, R. Wille, Conceptual clustering via convex-ordinal structures, in: O. Opitz, B. Lausen, R. Klar (Eds.), Information and Classification, Springer, Berlin, Heidelberg, 1993, pp. 85–98.

[38] G. Stumme, A. Mdche, FCA Merge: bottom-up merging of ontologies, in: Proceedings 17th International Conference on Artificial Intelligence (IJCAI'01), Seattle, WA, USA, 2001, pp. 225–230.

[39] G. Stumme, R. Taouil, Y. Bastide, L. Lakhal, Conceptual clustering with iceberg concept lattices, in: Proceedings of GI–Fachgruppentreffen Maschinelles Lernen'01, Universität Dortmund, vol. 763, Oktober 2001.

[40] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal, Fast computation of concept lattices using data mining techniques, in: Proceedings 7th International Workshop on Knowledge Representation Meets Databases, Berlin, 21–22 August 2000, CEUR-Workshop Proceeding. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/.

[41] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal, Intelligent structuring and reducing of association rules with formal concept analysis, in: Proceedings KI 2001 LNAI 2174, in: F. Baader, G. Brewker, T. Eiter (Eds.), KI 2001: Advances in Artificial Intelligence, Springer, Heidelberg, 2001, pp. 335–350.

[42] G. Stumme, R. Wille, U. Wille, Conceptual knowledge discovery in databases using formal concept analysis methods, in: Proceedings 2nd European Symposium on PKDD'98, in: J.M. Żytkow, in: M. Quafofou (Eds.), Principles of Data Mining and Knowledge Discovery, LNAI 1510, Springer, Heidelberg, 1998, pp. 450–458.

[43] G. Stumme, R. Wille (Eds.), Begriffliche Wissensverarbeitung—Methoden und Anwendungen, Springer, Heidelberg, 2000.

[44] F. Vogt, R. Wille, TOSCANA—A graphical tool for analyzing and exploring data, LNCS 894, Springer, Heidelberg, 1995, pp. 226–233.

[45] K. Waiyamai, R. Taouil, L. Lakhal, Towards an object database approach for managing concept lattices, in: Proceedings 16th International Conference on Conceptual Modeling, LNCS 1331, Springer, Heidelberg, 1997, pp. 299–312.

[46] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts, in: I. Rival (Ed.), Ordered Sets, Reidel, Dordrecht, Boston, 1982, pp. 445–470.

[47] R. Wille, Line diagrams of hierarchical concept systems, Int. Classif. 11 (1984) 77–86.

[48] R. Wille, Concept lattices and conceptual knowledge systems, Computers and Mathematics with Applications 23 (1992) 493–515.

[49] S. Wrobel, K. Morik, T. Joachims, Maschinelles lernen und data mining, in: G. Grz, C.-R. Rollinger, J. Schneeberger (Eds.), Handbuch der Knstlichen Intelligenz, vol. 3, Auflage, Oldenbourg, Munchen, Wien, 2000, pp. 517–597.

[50] A. Yahia, L. Lakhal, J.P. Bordat, R. Cicchetti, iO2: An algorithmic method for building inheritance graphs in object database design, in: Proceedings 15th International Conference on Conceptual Modeling, LNCS 1157, Springer, Heidelberg, 1996, pp. 422–437.

Gerd Stumme is Senior Researcher in the research group Knowledge Management at the Institute for Applied Informatics at the University of Karlsruhe. In his PhD at Darmstadt University of Technology he developed knowledge acquisition techniques based on Formal Concept Analysis. His recent research interests comprise knowledge discovery in databases, formal concept analysis, ontologies, semi-structured data, conceptual knowledge processing, and semantic web mining. Gerd Stumme is author of more than 40 publications and co-editor of a volume on Conceptual Knowledge Processing. In 2001, he was program co-chair of the Eighth International Conference on Conceptual Structures.

Rafik Taouil is a post-doctorate at INRIA Lorraine. He is currently working on algorithms for closed-set lattices and their applications for data mining and databases.

Yves Bastide has a PhD in Computer Science. He is studying algorithms and data structures for data mining, and applications on complex data.

Nicolas Pasquier received his PhD in Computer Science from the University Blaise Pascal, Clermont-Ferrand, France, in January 2000. He is now a faculty member, as Maître de Conférences, at the University of Nice-Sophia Antipolis and member of the Mecosi research project in the CNRS Laboratory of Computer Science, Signals and Systems of Sophia Antipolis in France.

His current research interests include database theory, database systems, data mining and knowledge discovery in very large databases. He received the Junior Researcher Award from the Inforsid association on information systems, databases and decision systems in 2000 for his contribution to the design of efficient association rule extraction methods.

Lotfi Lakhal received his PhD (1986) and the Habilitation for Directing Research (1992) from the University of Nice-Sophia Antipolis, France. He joined the Department of Mathematics and Computer Science of Blaise Pascal University, Clermont-Ferrand, France, and became a full professor in 1993. In 2000, he joined the Mediterranean University, IUT d'Aix-en-Provence, France. His research interests include statistical databases, database design, formal concept analysis, data mining, and data warehouses.