# Part II

# Closure Systems and Implications

# Chapter 4

# Closure systems

The algorithms that will be the central theme of our course were developed for concept lattices, but can be rephrased without reference to Formal Concept Analysis. The reason is that the algorithm essentially relies on a single property of concept lattices, namely that the set of concept intents is closed under intersections. The technique can be formulated for arbitrary intersection closed families of sets, that is, for *closure systems*. Readers who are familiar with closure systems but not with Formal Concept Analysis may prefer this approach.

But note that this means no generalization. We will show that closure systems are not more general than systems of concept intents.

## 4.1 Definition and examples

Closure systems occur frequently in mathematics and computer science. Their definition is very simple, but not very intuitive when encountered for the first time. The reason is their higher level of abstraction: closure systems are *sets of sets* with certain properties.

Let us recall some elementary notions how to work with sets of sets. For clarity, we shall normally use small latin letters for elements, capital latin letters for sets and calligraphic letters for sets of sets. Given a (nonempty) set $\mathcal{S}$ of sets, we may ask

- which elements occur in these sets? The answer is given by the **union** of $\mathcal{S}$, denoted by
$$\bigcup \mathcal{S} := \{x \mid \exists_{S\in\mathcal{S}} \ x \in S\}.$$
- which elements occur in each of these sets? The answer is given by the **intersection** of $\mathcal{S}$, denoted
$$\bigcap \mathcal{S} := \{x \mid \forall_{S\in\mathcal{S}} \ x \in S\}.$$

Some confusion with this definition is caused by the fact that a set of sets may (of course) be empty. Applying the above definition to the case $\mathcal{S} := \emptyset$ is no problem for the union, since

$$\bigcup \emptyset = \{x \mid \exists_{S\in\mathcal{S}} x \in S\} = \{x \mid \text{false}\} = \emptyset \ .$$

But it gives a problem for the intersection, because then the condition $\forall_{S\in\mathcal{S}} \ x \in S$ is fulfilled by *all* $x$ (because there is nothing to fulfill). But there is no *set of all* $x$; such sets are forbidden in set theory, because they would lead to contradictions.

For the case $\mathcal{S} = \emptyset$ the intersection is defined only with respect to some base set $M$. If we work with the subsets of some specified set $M$ (as we often do, for example with the set of all attributes of some formal context), then we define

$$\bigcap \emptyset := M.$$

A set $M$ with, say, $n$ elements, has $2^n$ subsets. The set of all subsets of a set $M$ is denoted $\mathfrak{P}(M)$ and is called the **power set** of the set $M$. To indicate that $\mathcal{S}$ is a set of subsets of $M$ we may therefore simply write $\mathcal{S} \subseteq \mathfrak{P}(M)$.

### 4.1.1 Closure systems

A closure system on a set $M$ is a set of subsets that contains $M$ and is closed under intersections. More formally,

**Definition 20** A **closure system** on a set $M$ is a set $\mathcal{C} \subseteq \mathfrak{P}(M)$ satisfying

- $M \in \mathcal{C}$, and
- if $\mathcal{D} \subseteq \mathcal{C}$, then $\bigcap \mathcal{D} \in \mathcal{C}$.

$\Diamond$

Closure systems are everywhere:

- The subtrees of any tree form a closure system, because the intersection of subtrees in any case is a subtree.

  That this is true can be seen as follows: recall that in a tree any two vertices are connected by a (unique) path. A set of vertices induces a subtree if any only if it contains with any two of its vertices also all vertices on the path between these two. Now consider any selection of subtrees and let $S$ be the set of vertices common to all these subtrees (i.e., their intersection). Let $v, w$ be two vertices in $S$. The path between $v$ and $w$ belongs to every subtree containing $v$ and $w$ and therefore to each of the selected subtree. It is therefore also contained in their intersection. Thus, $D$ is the vertex set of a subtree.

- Take any algebraic structure, for example a group or a vector space, and take the set of its subalgebras (subgroups, sub-spaces, resp.). This is a closure system, because the intersection of arbitrary subgroup is again a subgroup, the intersection of arbitrary subspaces is again a subspace, and, more generally, the intersection of subalgebras is a subalgebra.

  Similarly, consider the set $M := A^*$ of all (finite) words over some alphabet $A$. In other words, consider the free monoid over $A$. The set of its submonoids is a closure system, because any intersection of submonoids is closed under multiplication and contains the empty word.

- Or take the set of subintervals of the real interval $[0, 1]$, including the empty interval. Since any intersection of intervals is again an interval, we have a closure system.

  This example can be generalized to $n$ dimensions, where we obtain the closure system of convex sets.

- The set of downsets of any ordered set $(M, \leq)$ is a closure system on $M$. A **downset** (or **order ideal**) of $(M, \leq)$ is a subset $D \subseteq M$ such that whenever $d \in D$ and $m \in M$ with $m \leq d$, then $m \in D$. Dually, the set of all **order filters** (or **upsets**, respectively) is a closure system.

- Consider all **preorders** on some fixed base set $S$, that is, all transitive reflexive relations $R \subseteq S \times S$. Any intersection of transitive reflexive relations is again transitive and reflexive. Therefore, preorders form a closure system.

### 4.1.2 Closure operators

**Definition 21** A **closure operator** $\varphi$ on $M$ is a map assigning a **closure** $\varphi X \subseteq M$ to each set $X \subseteq M$, which is

**monotone:** $X \subseteq Y \Rightarrow \varphi X \subseteq \varphi Y$

**extensive:** $X \subseteq \varphi X$, and

**idempotent:** $\varphi\varphi X = \varphi X$.

(Conditions to be fulfilled for all $X, Y \subseteq M$.) $\Diamond$

For every example in the above list of closure systems we can also give a closure operator. In each of the following examples, $\varphi$ is a closure operator on the set $M$.

- Let $(M, E)$ be a tree with vertex set $M$ and let $\varphi$ be the mapping that maps each set $X$ of vertices to the vertex set of the smallest subtree containing $X$.
- For a group with carrier set $M$, define $\varphi X$ to be the subgroup generated by $X$.

  For a vector space with carrier set $M$, define $\varphi X$ to be the subspace generated by $X$.

  For a set $X$ of words over an alphabet $A$, let $\varphi X := X^*$ be the submonoid of $M := A^*$ which is generated by $X$.
- For any $X \subseteq M := [0, 1]$ let $\varphi X$ be the smallest interval containing $X$ (i.e., the convex closure of $X$).
- If $(M, \leq)$ is an ordered set then for any $X \subseteq M$ let

  $$\varphi X := \{m \in M \mid m \leq x \text{ for some } x \in X\}$$

  be the downset generated by $X$.
- For any relation $R$ on a set $S$, in other words, for any subset $R \subseteq S \times S =: M$, let $\varphi R$ denote the reflexive transitive closure of $R$.

The examples indicate why closure operators are so frequently met: their axioms describe the natural properties of a *generating process*. We start with some generating set $X$, apply the generating process and obtain the generated set, $\varphi X$, the closure of $X$. Such generating processes occur in fact in many different variants in mathematics and computer science.

### 4.1.3 The closure systems of intents and of extents

Closure systems and closure operators are closely related. In fact, there is a natural way to obtain from each closure operator a closure system and vice versa. It works as follows:

**Lemma 10** *For any closure operator, the set of all closures is a closure system. Conversely, given any closure system $\mathcal{C}$ on $M$, there is for each subset $X$ of $M$ a unique smallest set $C \in \mathcal{C}$ containing $X$. Taking this as the closure of $X$ defines a closure operator. The two transformations are inverse to each other.*

**Proof** Exercise 1. $\Box$

Thus closure systems and closure operators are essentially the same. We can add to this:

**Theorem 11** *A closure system $\mathcal{C}$ on a set $M$ can be considered as a complete lattice, ordered by set inclusion $\subseteq$. The infimum of any subfamily $\mathcal{D} \subseteq \mathcal{C}$ is equal to $\bigcap \mathcal{D}$, and the supremum is the closure of $\bigcup \mathcal{D}$. Conversely we can find for any complete lattice $L$ a closure system that is isomorphic to $L$.*

**Proof** Exercise 2. □

So closure systems and complete lattices are also very closely related.

It comes as no surprise that concept lattices can be subsumed under this relationship. It follows from the Basic Theorem (Thm. 4 (p. 25)) that the set of all concept intents of a formal context is closed under intersections and thus is a closure system on $M$. Dually, the set of all concept extents always is a closure system on $G$. The corresponding closure operators are just the two operators

$$X \mapsto X''$$

on $M$ and $G$, respectively.

Conversely, given any closure system $\mathcal{C}$ on a set $M$, we can construct a formal context such that $\mathcal{C}$ is the set of concept intents. It can be concluded from the Basic Theorem that for example $(\mathcal{C}, M, \ni)$ is such a context. In particular, whenever a closure operator on some set $M$ is considered, we may assume that it is the closure operator

$$A \mapsto A''$$

on the attribute set of some formal context $(G, M, I)$.

Thus, closure systems and closure operators, complete lattices, systems of concept intents, and systems of concept extents: all these are very closely related. It is not appropriate to say that they are "essentially the same", but it is true that all these structures have the same degree of expressiveness; none of them is a generalization of another. A substantial result proved for one of these structures can usually be transferred to the others, without much effort.

## 4.2 Formal contexts in Computer Science: examples

We have seen that closure systems can be represented by formal contexts, and this is often a very convenient way of handling a closure system. It is often beneficial to ask if a given closure system can nicely be described by a formal context.

On the other hand, formal contexts occur in mathematics and computer science without being named that way. It is no surprise that in such situations closure operators and complete lattices can be introduced.

### 4.2.1 Ontology Learning.

Ontologies are "explicit specification[s] of a conceptualization" [Gr94]. 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 [MaS00]) 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 [SM01], this approach has been used in FCA–Merge, a technique for supporting the merging of ontologies. The 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 ontology.

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.

### 4.2.2 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 [KS94], the use of Formal Concept Analysis for analyzing the configuration space of C++ programs is discussed. 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. Decompositions of the concept lattice indicate possibilities for re-facturing the code.

### 4.2.3 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 [ST98], 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.

### 4.2.4 Model classes and theories, equational classes and equational theories

Consider propositional formulae over a set $X$ of variables. The potential models of such formulae are truth value assignments $\varepsilon : X \to \{\text{TRUE}, \text{FALSE}\}$. To express that a certain $\varepsilon$ is a model of a formula $f$, one writes $\varepsilon \models f$. Denoting by $\mathcal{F}(X)$ the set of all formulae over $X$ and by $\mathcal{M}(X) := \{\text{TRUE}, \text{FALSE}\}^X$ the set of all truth value assignments, we can define the formal context

$$(\mathcal{M}(X), \mathcal{F}(X), \models).$$

Given any subset $A \subseteq \mathcal{M}(X)$ of assignments, we may ask which formulae hold in all these; the set of these formulae is the **propositional theory** of $A$. Conversely, for each set $F \subseteq \mathcal{F}$ of formulae we have the set of assignments for which all these formulae hold; these are the **models**[1] of $F$. These two mappings

$$\begin{aligned}
\mathrm{Th} : \mathfrak{P}(\mathcal{M}(X)) \to \mathfrak{P}(\mathcal{F}(X)), \qquad & A \mapsto \{f \in \mathcal{F}(X) \mid a \models f \quad \forall\, a \in A\} \\
\mathrm{Mod} : \mathfrak{P}(\mathcal{F}(X)) \to \mathfrak{P}(\mathcal{M}(X)), \qquad & F \mapsto \{m \in \mathcal{M}(x) \mid m \models f \quad \forall\, f \in F\}
\end{aligned}$$

are of course the derivation operators of the context $(\mathcal{M}(X), \mathcal{F}(X), \models)$. We obtain two closure operators, one on $\mathcal{M}(X)$ and one on $\mathcal{F}(X)$. They are just the two closure operators

$$X \mapsto X''$$

associated to the derivation operators.

$$A \mapsto \mathrm{Mod}(\mathrm{Th}(A)) \qquad \text{for } A \subseteq \mathcal{M}(X)$$

maps each set of truth value assignments to itself, because each model set can completely be described by its theory;

$$F \mapsto \mathrm{Th}(\mathrm{Mod}(F)) \qquad \text{for } F \subseteq \mathcal{F}(X)$$

extends each set $F$ of formulae by those formulae that follow (semantically) from $F$ (to the theory generated by $F$).

The formal concepts of this formal context have as intents just the propositional theories and as extents their model classes.

A very similar example can be obtained for (universal) algebras of a fixed signature, except that in that case we have to introduce some restrictions in order to avoid proper classes (classes that are not sets). Writing

$$\underline{A} \models e$$

to express that the equation $e$ holds in the algebra $\underline{A}$, we obtain, as above, model classes ("varieties") and theories ("equational theories").

### 4.2.5 Equivalence relations

A binary relation $\Theta \subseteq S \times S$ on a set $S$ is called an **equivalence relation** if it is reflexive, transitive, and symmetric. It is easy to check that these properties are preserved under arbitrary intersections. Therefore the set of all evivalence relations on $S$ is a closure system. The associated closure operator takes as input any relation $R \subseteq S \times S$ and outputs its reflexive, symmetric, transitive closure, that is, the smallest eqivalence relation containing $R$.

It is easy to describe the supremum- and the infimum-irreducible equivalence relations in terms of their equivalence classes:

- An equivalence relation is supremum-irreducible iff all its classes, except one, have only one element and the exceptional class has exactly teo elements.
- An equivalence relation is infimum-irreducible iff it has exactly two equivalence classes.

A formal context can be defined as

$$\left(\binom{S}{2}, \mathfrak{P}(S \setminus \{s\}) \setminus \{\emptyset\}, I\right), \qquad \text{(for some fixed } s \in S\text{)}$$

[1]A truth value assignment $\varepsilon$ can be specified by giving the set $T \subseteq X$ of those variables to which the value TRUE is assigned:

$$T := \{x \in X \mid \varepsilon(x) = \text{TRUE}\}.$$

*Models* of a propositional theory can therefore be identified with certain subsets of the set $X$.

where $\binom{S}{2}$ denotes the set of all two-element subsets of $S$, and for $a, b \in S$, $C \subseteq S \setminus \{s\}$,

$$\{a, b\}\ I\ C :\iff |\{a, b\} \cap C| \neq 1.$$

The concept extents of this formal context are precisely all equivalence relations on $S$.

### 4.2.6 Orders, order filters, order ideals

From an arbitrary ordered set $(P, \leq)$ we can obtain several interesting parts, for example its *order ideals*, its *order filters*, and its *cuts*. Order filters and -ideals have been introduced above (on page 46). A **cut** of $(P, \leq)$ is a pair $(A, B)$ of subsets $A, B \subseteq P$ such that

- $A$ is the set of all lower bounds of $B$, and
- $B$ is the set of all upper bounds of $A$.

From an ordered set we can also derive several formal contexts. It is not difficult to show that

- The cuts of $(P, \leq)$ are precisely the formal concepts of the context $(P, P, \leq)$.
- The order ideals of $(P, \leq)$ are precisely the concept extents of the formal context $(P, P, \not\geq)$.
- The order filters of $(P, \leq)$ are precisely the concept intents of the formal context $(P, P, \not\geq)$. In fact, $(A, B)$ is a formal concept of $(P, P, \not\geq)$ iff $A$ is a downset, $B$ is an upset and $A = P \setminus B$.

### 4.2.7 Bracketings and permutations

## 4.3 The Next Closure algorithm

We present a simple algorithm that solves the following task: For a given closure operator an a finite set $M$, it computes all closed sets.

There are many ways to achieve this. Our algorithm is particularly simple. We shall discuss efficiency considerations below. It also allows many useful modifications, some of which will be used in our more advanced applications. We start with the simplest version.

### 4.3.1 Representing sets by bit vectors

We start by giving our base set $M$ an arbitrary linear order, so that

$$M = \{m_1 < m_2 < \cdots < m_n\},$$

where $n$ is the number of elements of $M$. Then every subset $S \subseteq M$ can conveniently be described by its **characteristic vector**

$$\varepsilon_S : M \to \{0, 1\},$$

given by

$$\varepsilon_S(m) := \begin{cases} 1 & \text{if } m \in S \\ 0 & \text{if } m \notin S \end{cases}.$$

For example, if the base set is

$$M := \{a < b < c < d < e < f < g\},$$

then the characteristic vector of the subset $S := \{a, c, d, f\}$ is 1011010. In concrete examples we prefer to write a cross instead of a 1 and a blank or a dot instead of a 0, similarly as in the cross tables representing formal contexts. The characteristic vector of the subset $S := \{a, c, d, f\}$ will therefore be written as

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| × | . | × | × | . | × | . |

.

In this notation it is easy to see if a given set is a subset of another given set, etc.

The set $\mathfrak{P}(M)$ of all subsets of the base set $M$ is naturally ordered by the subset–order $\subseteq$. This is a complete lattice order, and $(\mathfrak{P}(M), \subseteq)$ is called the **power set lattice** of $M$. The subset-order is a *partial order.* We can also introduce a *linear* or *total* order of the subsets, for example the **lexicographic** or **lectic order** $\leq$, defined as follows: Let $A, B \subseteq M$ be two distinct subsets. We say that $A$ is *lectically smaller* than $B$, if the smallest element in which $A$ and $B$ differ belongs to $B$. Formally,

$$A < B \quad :\Longleftrightarrow \quad \exists_i \, (i \in B, \, i \notin A, \, \forall_{j<i} \, (j \in A \iff j \in B)).$$

For example $\{a, c, e, f\} < \{a, c, d, f\}$, because the smallest element in which the two sets differ is $D$, and this element belongs to the larger set. This becomes even more apparent when we write the sets as vectors and interprete them as binary numbers:

$$\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ & & & \updownarrow & & & \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array} .$$

Note that the lectic order extends the subset-order, i.e.,

$$A \subseteq B \Rightarrow A \leq B.$$

The following notation is helpful:

$$A <_i B \quad :\Longleftrightarrow \quad (i \in B, \, i \notin A, \, \forall_{j<i} \, (j \in A \iff j \in B)).$$

In words: $A <_i B$ iff $i$ is the smallest element in which $A$ and $B$ differ, and $i \in B$.

**Proposition 12** *1. $A < B$ if and only if $A <_i B$ for some $i \in M$.*

*2. If $A <_i B$ and $A <_j C$ with $i < j$, then $C <_i B$.*

### 4.3.2 Closures in lectic order

We consider a closure operator

$$A \mapsto A''$$

on the base set $M$. To each subset $A \subseteq M$ it gives[2] its closure $A'' \subseteq M$. Our task is to find a list of all these closures. In principle we might just follow the definition, compute for each subset $A \subseteq M$ its closure $A''$ and include that in the list. The problem is that different subsets may have identical closures. So if we want a list that contains each closure *exactly once*, we will have to check many times if a computed closure already exists in the list. Moreover, the number of subsets is exponential: a set with $n$ elements has $2^n$ subsets. The naive algorithm "for each $A \subseteq M$, compute $A''$ and check if the result is already listed" therefore requires an exponential number of lookups in a list that may have exponential size.

A better idea is to generate the closures in some predefined order, thereby guaranteeing that every closure is generated only once. The reader may guess that we

[2]For our algorithm it is not important *how* the closure is computed.

shall generate the closures in lectic order. We will show how to compute, given a closed set, the *lectically next* one. Then no lookups are necessary. Actually, it will not even be necessary to store the list. For many applications it will suffice to generate the list elements on demand. Therefore we do not have to store exponentially many closed sets. Instead, we shall store just *one*!.

To find the next closure we define for $A \subseteq M$ and $m_i \in M$

$$A \oplus m_i := ((A \cap \{m_1, \dots, m_{i-1}\}) \cup \{m_i\})''.$$

We illustrate this definition by an example: Let $A := \{a, c, d, f\}$ and $m_i := e$.

$$\begin{array}{|c|c|c|c|c|c|c|}\multicolumn{4}{c}{} & \multicolumn{1}{c}{\downarrow} & \multicolumn{2}{c}{} \\ \hline \times & . & \times & \times & . & \times & . \\ \hline \end{array}.$$

We first remove all elements that are greater or equal $m_i$ from $A$:

$$\begin{array}{|c|c|c|c|c|c|c|}\multicolumn{4}{c}{} & \multicolumn{1}{c}{\downarrow} & \multicolumn{2}{c}{} \\ \hline \times & . & \times & \times & . & . & . \\ \hline \end{array}.$$

Then we insert $m_i$

$$\begin{array}{|c|c|c|c|c|c|c|}\multicolumn{4}{c}{} & \multicolumn{1}{c}{\downarrow} & \multicolumn{2}{c}{} \\ \hline \times & . & \times & \times & \times & . & . \\ \hline \end{array}$$

and form the closure. Since we have not yet specified the closure operator $\cdot''$ (i. e., we have not given a formal context), the example stops here with

$$A \oplus e = \{a, c, d, e\}''.$$

**Proposition 13** *1. If $i \notin A$ then $A < A \oplus i$.*

*2. If $B$ is closed and $A <_i B$ then $A \oplus i \subseteq B$, in particular $A \oplus i \leq B$.*

*3. If $B$ is closed and $A <_i B$ then $A <_i A \oplus i$.*

**Theorem 14** *The smallest closed set larger than a given set $A \subset M$ with respect to the lectic order is*

$$A \oplus i,$$

*$i$ being the largest element of $M$ with $A <_i A \oplus i$.*

Now we are ready to give the algorithm for generating all extents of a given context $(G, M, I)$: The lectically smallest extent is $\emptyset''$. For a given set $A \subset G$ we find the lectically next extent by checking all elements $i$ of $G \setminus A$, starting from the largest one and continuing in a descending order until for the first time $A <_i A \oplus i$. $A \oplus i$ then is the "next" extent we have been looking for. These three steps are made explicit in Figures 4.1 to 4.3.

```
Algorithm First Closure
Input:  A closure operator $X \mapsto X''$ on a finite set $M$.
Output:  The closure $A$ of the empty set.
begin
  $A := \emptyset''$;
end.
```

Figure 4.1: First Closure.

```
Algorithm Next Closure
Input:  A closure operator $X \mapsto X''$ on a finite set $M$,
        and a subset $A \subseteq M$.
Output: $A$ is replaced by the lectically next closed set.
begin
  $i$ := largest element of $M$;
  $i$ := succ($i$);
  success := false;
  repeat
    $i$ := pred($i$);
    if $i \notin A$ then
    begin
      $A := A \cup \{i\}$;
      $B := A''$;
      if $B \setminus A$ contains no element $< i$ then
      begin
        A:= B;
        success := true;
      end;
    end else $A := A \setminus \{i\}$;
  until erfolg or $i =$ smallest element of $M$.
end.
```

Figure 4.2: Next Closure.

```
Algorithm All Closures
Input:  A closure operator $X \mapsto X''$ on a finite set $M$.
Output: All closed sets in lectic order.
begin
  First_Closure;
  repeat
    Output $A$;
    Next_Closure;
  until not success;
end.
```

Figure 4.3: Generating all closed sets.

There are several implementations of this algorithm. The best-known is probably the program ConImp by Peter Burmeister, which is particularly common on DOS-computers. For the world of Unix there is a version named Concepts by Christian Lindig. Both programs are at present available for non-commercial purposes.[3]

### 4.3.3 Examples

——–(to be written)——–

[3]e.g. free of charge via the Internet:
ftp.mathematik.th-darmstadt.de:/pub/department/software/conceptanalysis
or ftp.ips.cs.tu-bs.de:/pub/local/softech/misc.

### 4.3.4 The number of concepts may be exponential

The problem of computing concept lattices has exponential worst-case complexity: The context $\mathbb{K} := (\{1,\dots,n\},\{1,\dots,n\},\neq)$ has $n$ objects and $n$ attributes, while its concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$ has $2^n$ concepts. Therefore all algorithms necessarily have an exponential worst-case complexity. However, it is of interest to analyze the situation in more detail.

### 4.3.5 Computation time per concept is polynomial

———(to be written)———

## 4.4 Iceberg Lattices and Titanic

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 and for improving the response times of algorithms for mining association rules. We will discuss association rules later in Section .

In this section we show that, vice versa, FCA can also benefit from ideas used for mining association rules by presenting a second, efficient algorithm for computing concept lattices, 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.

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 —, and as we will see later, as a condensed representation of frequent itemsets, as a base of association rules, and as a visualization tool for association rules.

### 4.4.1 Iceberg Concept Lattices

In the worst case, the size of concept lattices grows exponentially with the size of the context. Hence for most applications one has to consider strategies for dealing with such large concept lattices.

In this section, we present an approach based on *frequent itemsets* as known from data mining [AIS93]: 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 22** 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 $\operatorname{supp}(B) := \frac{|B'|}{|G|}$. $B$ is said to be a *frequent* attribute set if $\operatorname{supp}(B) \geq \text{minsupp}$.

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 $\mathbb{K}$. ◊

**Lemma 15** *For all $B \subseteq M$, we have* $\operatorname{supp}(B) = \operatorname{supp}(B'')$.

**Proof** By applying Proposition 1, we obtain $\operatorname{supp}(B) = \frac{|B'|}{|G|} = \frac{|B'''|}{|G|}\operatorname{supp}(B'')$ □

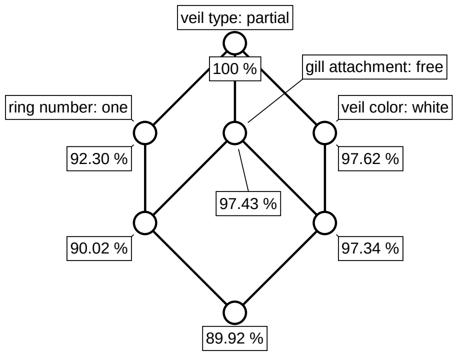

Figure 4.4: Iceberg concept lattice of the mushroom database with minsupp = 85 %

Because the support function is monotonously decreasing (i. e., $B_1 \subseteq B_2 \Rightarrow \text{supp}(B_1) \geq \text{supp}(B_2)$), 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's have a closer look to iceberg concept lattices:

**Example 4** As running example, we use the Mushroom database from the *UCI KDD Archive* (`http://kdd.ics.uci.edu/`). It consists of a database with 8,416 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 8,416 objects and 80 attributes. 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: Figure 4.4 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-frequent 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.

In the diagram, we can detect the implication

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

It means that all mushrooms with one ring and a white veil have free gills. Implications are discussed in more detail in the next chapter. This implication is indicated in the diagram 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 %). It is globally valid in the Mushroom database, 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. Figure 4.5 shows the Mushroom iceberg concept lattice for a minimum support of

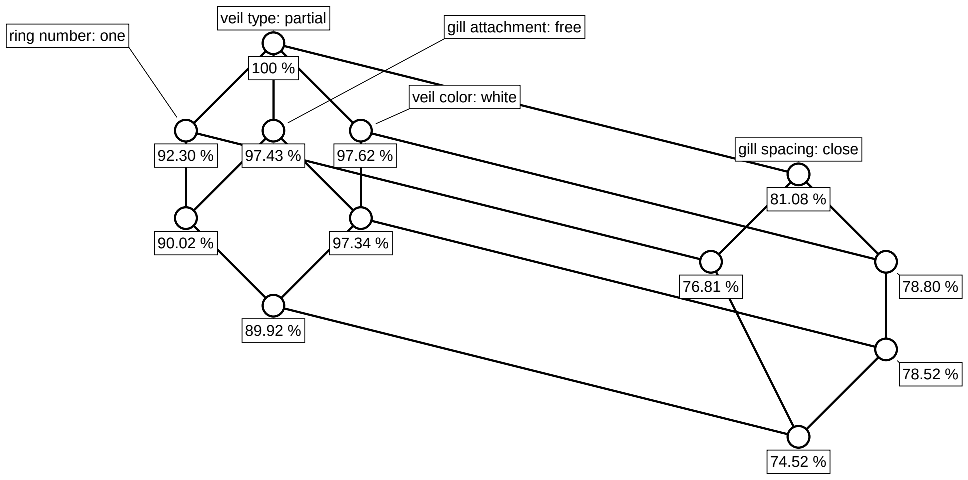

Figure 4.5: Iceberg concept lattice of the mushroom database with minsupp = 70 %

Table 4.1: 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 | $2^{80}$ |

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-frequent attribute 'gill spacing: close' (having 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. Figure 4.6 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 4.1, 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.

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: Figure 4.7 shows the iceberg concept lattice as a nested line diagram. The diagram provides exactly the same information than Figure 4.6, but in a more

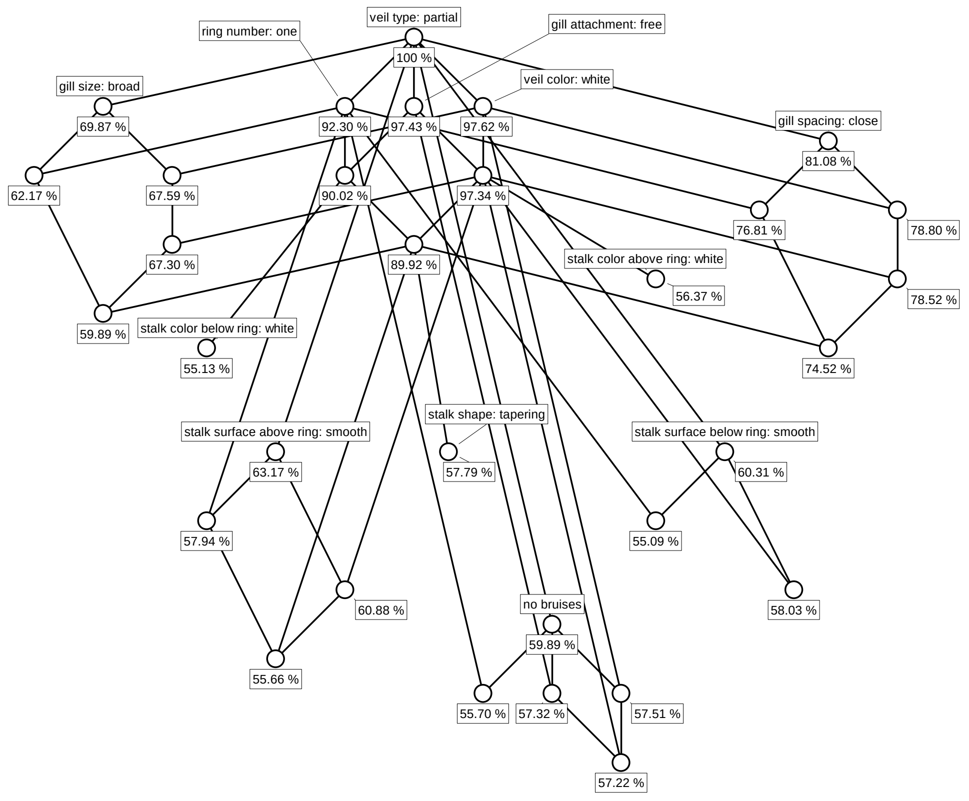

Figure 4.6: Iceberg concept lattice of the mushroom database with minsupp = 55 %

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

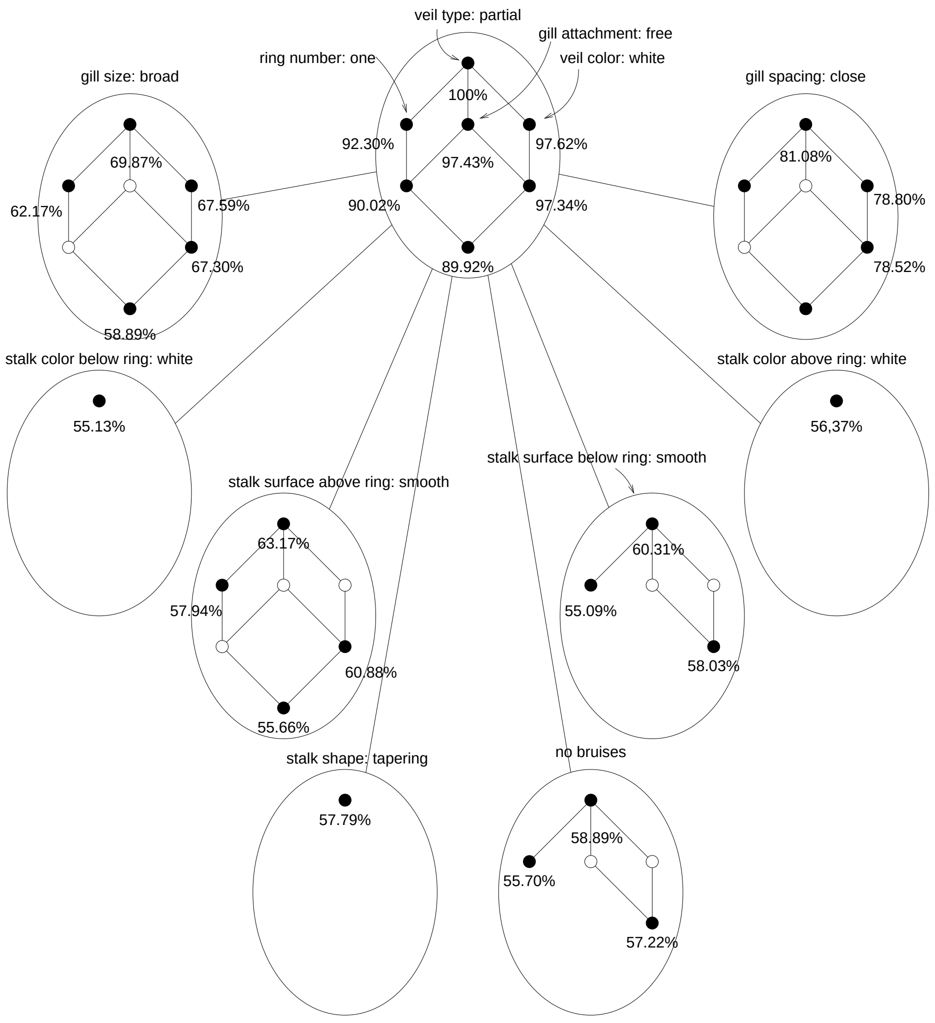

Figure 4.7: Nested line diagram of the iceberg concept lattice in Figure 4.6

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 Section 2.4.2, 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 Figure 4.7, the inner diagram contains the top-most attributes, the outer diagram the next-most attributes. The resulting diagram shows the most important 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*, which is a specific technique for *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. 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 1990ies on.

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., [CR93]), vertical reduction by conceptual scaling (see Chapter3, 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:

**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 $\text{supp}(B) = \text{supp}(B'')$, 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 [BTPSL00] which is related to TITANIC, and which efficiently computes frequent itemsets.

**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 [BPTSL00] and [STBPL01], 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.

**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., association rules with less than 100 % confidence), which is presented in [STBPL01], can also be visualized directly in the line diagram of an iceberg concept lattice. The Luxenburger basis is derived from [Lu91]. 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. Figure 4.8

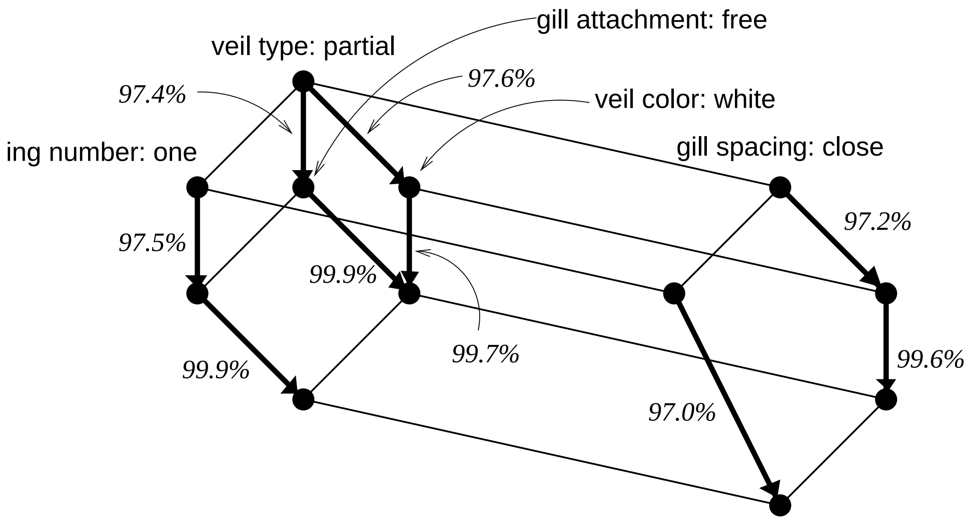

Figure 4.8: Visualization of the Luxenburger basis for minsupp $= 70\,\%$ and minconf$= 95\,\%$

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} $\rightarrow$ {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 Figure 4.5. 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 [STBPL01]. In comparison with other visualization techniques for association rules, 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.

### 4.4.2 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.

Section 4.1.3 shows 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 $\mathcal{H}_h$ related to the closure operator $h$ when there exists a *weight function* compatible with the closure operator:

**Definition 23** A *weight function* on $\mathfrak{P}(M)$ is a function $s\colon \mathfrak{P}(M) \rightarrow P$ from the powerset of $M$ to a totally ordered set $(P, \leq)$ 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

$(i)$ $X \subseteq Y \;\Rightarrow\; s(X) \geq s(Y)$,

$(ii)$ $h(X) = h(Y) \;\Rightarrow\; s(X) = s(Y)$,

$(iii)\ X \subseteq Y \wedge s(X) = s(Y) \Rightarrow h(X) = h(Y)$ .

$\diamond$

*Remark.* In the sequel, we will consider $(P, \leq)$ 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 $X \subseteq Y \Rightarrow s(X) \leq s(Y)$ 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 $\mathcal{H}_h$ related to the closure operator $h$ by using the weight function $s$.

### 4.4.3 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.

#### 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 16** *Let $X \subseteq M$. Then*

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

**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 23. 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 23. Hence $m \in h((X \cup \{m\})) = h(X)$. $\square$

Hence if we know the weights of all sets, then we can compute the closure operator ($\rightarrow$ Algorithm 3, steps 3–7).[4] In the next subsection we discuss for which sets it is necessary to compute the closure in order to obtain all closed sets. In Subsection 4.4.3 we discuss how the weights needed for those computations can be determined.

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

### A level-wise approach for computing all closed sets

One can now compute the closure system $\mathcal{H}$ by applying Proposition 16 to all subsets $X$ of $M$. But this is not efficient, since many closed sets will be determined several times.

**Definition 24** We define an equivalence relation $\theta$ 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 $[X] := \{Y \subseteq M \mid (X,Y) \in \theta\}$. $\Diamond$

If we knew the equivalence relation $\theta$ 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.

**Definition 25** A *$k$-set* is a subset $X$ of $M$ with $|X| = k$. For $\mathcal{X} \subseteq \mathfrak{P}(M)$, we define $\mathcal{X}_k := \{X \in \mathcal{X} \mid X \text{ is } k\text{-set}\}$. For $\mathcal{X} = \mathfrak{P}(M)$, we also write $\mathfrak{P}_k(M)$ for $\mathcal{X}_k$. $\Diamond$

At the $k$th 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 26** 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 $\mathcal{K}$. $\Diamond$

We have $\mathcal{H} = \{h(X) \mid X \in \mathcal{K}\}$, because there is at least one key set in each equivalence class of $\theta$. 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 $\mathfrak{P}(M)$ level-wise:

**Proposition 17** *The set $\mathcal{K}$ is an order ideal of $(\mathfrak{P}(M), \subseteq)$; i. e., $Y \in \mathcal{K}$ and $X \subseteq Y$ implies $X \in \mathcal{K}$, for all $X, Y \subseteq M$.*

**Proof** Let $X \subseteq Y$ and $X$ be a non-key set. Then there exists a minimal $Z \in [X]$ with $Z \subset X$.[5] From $h(Z) = h(X)$ it follows that $h(Y) = h(Y \setminus (X \setminus Z))$. Hence $Y$ is not minimal in $[Y]$ and thus by definition not a key set. $\Box$

The definition of an order ideal is equivalent to $X \notin \mathcal{K},\ X \subseteq Y \Rightarrow Y \notin \mathcal{K}$, 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 [AS94], 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 27** Let $\mathcal{I}$ be an order ideal of $\mathfrak{P}(M)$. A *candidate set* for $\mathcal{I}$ is a subset of $M$ such that all its proper subsets are in $\mathcal{I}$. $\Diamond$

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.

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

**Lemma 18** *Let $\mathcal{X} \subseteq \mathfrak{P}_k(M)$, and let $\mathcal{Y}$ be the set of all candidate $(k+1)$-sets for the order ideal $\downarrow\mathcal{X} := \{Y{\in}\mathfrak{P}(M) \mid \exists X{\in}\mathcal{X}{:}\, Y{\subseteq}X\}$ (i. e., the order ideal generated by $\mathcal{X}$).*

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

**Proof** *1.* Let $\mathcal{I} := (\downarrow\mathcal{X}) \cup \mathcal{Z}$. Let $Y \in \mathcal{I}$ and $X \subset Y$. We have to show that $X \in \mathcal{I}$. If $Y \in \downarrow\mathcal{X}$ then $X \in \downarrow\mathcal{X} \subseteq \mathcal{I}$ because $\downarrow\mathcal{X}$ is an order ideal. If $Y \in \mathcal{Z}$ then $X \in \downarrow\mathcal{X} \subseteq \mathcal{I}$ by Definition 27.

*2.* Suppose that there exists $Y \in \mathcal{I}_{k+1} \setminus \mathcal{Y}$. As $Y \notin \mathcal{Y}$, there exists $X \subset Y$ with $|X| = k$ and $X \notin \mathcal{I}_k$. Hence $Y \notin \mathcal{I}_{k+1}$. 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 19** *Let $\mathcal{X} \subseteq \mathfrak{P}_{k-1}(M)$. Let*

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

*and*

$$\mathcal{C} := \left\{X \in \widetilde{\mathcal{C}} \mid \forall x \in X{:}\, X \setminus \{x\} \in \mathcal{X}\right\} \quad .$$

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

**Proof** The definition of $C$ is equivalent to $C := \{x \in \widetilde{C} \mid X$ is candidate set for $\downarrow\mathcal{X}\}$. 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 $x_1 < \ldots < x_k$. Since $X$ is a candidate set, all its proper subsets are in $\downarrow\mathcal{X}$ — 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 $\mathcal{X}$. Hence $X \in \mathcal{I}$ by definition of $\widetilde{C}$. □

This generation procedure was first used in the Apriori algorithm [AS94] for the specific case of frequent itemsets.

Unlike in the Apriori algorithm, 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 [AS94]. Based on this additional step, at each iteration the non-key sets among the candidate sets are pruned (→ Algorithm 1, step 8) by using the second part of the following proposition.

**Proposition 20** *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 $s(X) \neq \min\{s(X \setminus \{m\}) \mid m \in X\}$.*

**Proof** *1.* The "if" part follows from Definition 23 (*iii*), the "only if" part from Definition 23 (*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 $s(X) = \min\{s(X \setminus \{m\}) \mid m \in X\}$ holds.

**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 21** *If $X$ is not a key set, then*

$$s(X) = \min\{s(K) \mid K \in \mathcal{K}, K \subseteq X\} \ .$$

**Proof** "$\geq$": Let $K$ be a key set with $K\theta X$ and $K \subseteq X$. Then $s(X) = s(K) \geq \min\{s(K) \mid K \in \mathcal{K}, K \subseteq X\}$.

"$\leq$": Suppose that there exists $K \in \mathcal{K}$ with $K \subseteq X$ and $s(K) < s(X)$. Then $K \not\subseteq X$ by Definition 23 $(i)$. Contradiction. □

Hence it is sufficient to compute the weights of the candidate sets only (by calling a function depending on the specific application $\rightarrow$ 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.

### 4.4.4 The Titanic Algorithm

The pseudo-code is given in Algorithm 1. A list of notations is provided in Table 4.2.

**Algorithm 1** Titanic

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

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+5).

In later iterations, the candidate $k$-sets are determined by the function Titanic-Gen (step 12 $\rightsquigarrow$ Algorithm 2) which is (except step 5) a straight-forward implementation of Proposition 19. The 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 20(2).

Once the candidate $k$-sets are determined, the function Weigh($\mathcal{X}$) is called to compute, for each $X \in \mathcal{X}$, 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 $X \in \mathcal{X}$ with *a single pass* of the context (see Section 4.4.5).

Table 4.2: Notations used in TITANIC

| | |
|---|---|
| $k$ | is the counter which indicates the current iteration. In the $k$th iteration, all key $k$-sets are determined. |
| $\mathcal{K}_k$ | contains after the $k$th iteration all key $k$-sets $K$ together with their weight $K.s$ and their closure $K$.closure. |
| $\mathcal{C}$ | 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. |

**Algorithm 2** TITANIC-GEN

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets $K$ with their weight $K.s$.

Output: $\mathcal{C}$, the set of candidate $k$-sets $C$
with the values $C.p\_s := \min\{s(C \setminus \{m\} \mid m \in C\}$.

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

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.

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 $\leadsto$ Algorithm 3). The CLOSURE function (Algorithm 3) is a straight-forward implementation of Proposition 16 (steps 3–7) and Proposition 21 (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 20 (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 the next section.

### 4.4.5 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 23 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.

**Algorithm 3** CLOSURE($X$) for $X \in \mathcal{K}_{k-1}$

```
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 $X \cup \{m\} \in \mathcal{C}$ then $s \leftarrow (X \cup \{m\}).s$
5)          else $s \leftarrow \min\{K.s \mid K \in \mathcal{K},\ K \subseteq X \cup \{m\}\}$;
6)    if $s = X.s$ then $Y \leftarrow Y \cup \{m\}$
7) end;
8) return $Y$.
```

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 4.4.6), only the WEIGH function has to be adapted.

### Computation of Concept Lattices

In the following, we will use the closure operator $B \mapsto B''$, for $B \subseteq M$. As said before, it is a closure operator on $M$. The related closure system (i. e., the set of all $B \subseteq M$ with $B'' = B$) is 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 22. We show first that the support function fulfills the conditions of Definition 23:

**Lemma 22** *Let $X, Y \subseteq M$.*

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

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

$$\text{supp}(Y) = \frac{|Y'|}{|G|} \leq \frac{|X'|}{|G|} = \text{supp}(X).$$

*2.* $X\,\theta\,Y \iff X'' = Y'' \iff X''' = Y''' \iff X' = Y' \Rightarrow$

$$s(X) = \frac{|X'|}{|G|} = \frac{|Y'|}{|G|} = s(Y)\ .$$

*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 23** *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

   $$1')\quad \emptyset.s \leftarrow 1$$

   since we know that $\text{supp}(\emptyset) = 1$. We avoid one call of the WEIGH function.

**Algorithm 4** The WEIGH algorithm for concept lattices

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

| | 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 | | × | × | | |

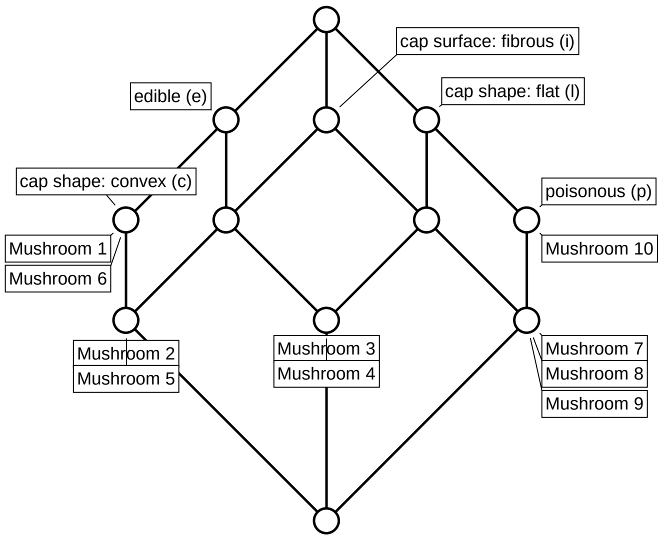

Figure 4.9: Example for the TITANIC algorithm

2. For concept lattices, WEIGH determines the weights — that is, the supports — of all $X \in \mathcal{X}$ with a single pass over the context. This is (together with the fact that only $\max\{|X| \mid X \subseteq M$ 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$(Y, \mathcal{X})$ returns, for $Y \subseteq M$ and $\mathcal{X} \subseteq \mathfrak{P}(M)$, all $X \in \mathcal{X}$ with $Y \subseteq X$. It uses a tree structure with hash tables (as described in [PBTL98]) to efficiently encode $\mathcal{X}$.

**Example 5** For explaining how TITANIC works, we will use the mushroom example again, but will reduce it to the first ten objects, and to the first five attributes (see Figure 4.9).

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

<u>$k = 0$:</u>

| step 1 | | step 2 |
|---|---|---|
| $X$ | $X.s$ | $X \in \mathcal{K}_k$? |
| $\emptyset$ | 1 | yes |

<u>$k = 1$:</u>

| 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:
<u>$k = 2$:</u>

| 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 $\leftarrow \{i\}$

<u>$k = 3$:</u>

| 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 $\leftarrow \{l,i\}$

<u>$k = 4$:</u>

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 Figure 4.9). The algorithm determined the support of $5 + 10 + 3 = 18$ attribute sets in three passes of the database.

**Algorithm 5** TITANIC improved for iceberg concept lattices

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

**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. The next lemma presents its closure operator.

**Lemma 24** *Let* $\mathbb{K} := (G, M, I)$ *be a context, and let* minsupp $\in [0, 1]$. *The set* $\mathcal{F} := \{B \subseteq M \mid (A, B) \in \underline{\mathfrak{B}}(\mathbb{K}), \text{supp}(B) \geq \text{minsupp}\} \cup \{M\}$ *is a closure system on* $M$. *Its closure operator is given by* $h(X) := X''$ *if* $\text{supp}(X) \geq \text{minsupp}$ *and* $h(X) := M$ *else. The weight function* $s(X) := \text{supp}(X)$ *if* $\text{supp}(X) \geq \text{minsupp}$ *and* $s(X) := -1$ *else is compatible with the closure operator.*

**Proof** $\widetilde{\mathcal{F}} := \{B \subseteq M \mid \text{supp}(B) \geq \text{minsupp}\} \cup \{M\}$ is a closure system, since it is closed under arbitrary intersections. $\text{Int}(\mathbb{K}) := \{B \subseteq M \mid (A, B) \in \underline{\mathfrak{B}}(\mathbb{K})\}$ is a closure system as shown in Section 4.1.3. 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 4.4.4 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 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 4.4.5.
- 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** TITANIC-GEN for iceberg concept lattices

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets $K$ with their support $K.s$.

Output: $\mathcal{C}$, the set of candidate $k$-sets $C$ with the values
$C.p\_s := \min\{s(C \setminus \{m\} \mid m \in C\}$.

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

**Algorithm 7** CLOSURE for iceberg concept lattices

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

- 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 24.

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

**Example 6** Although TITANIC only needs three passes of the database to compute the iceberg lattice in Figure 4.4 (and four passes for the one in Figure 4.6), 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 Figure 4.9, 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 5:

<u>$k = 0$:</u>

| step 1 | | step 2 |
|---|---|---|
| $X$ | $X.s$ | $X \in \mathcal{K}_k$? |
| $\emptyset$ | 1 | yes |

<u>$k = 1$:</u>

| 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 $k = 2$. Here, the first infrequent sets are reached:

<u>$k = 2$:</u>

| step 12 | | step 7 | step 9 |
|---|---|---|---|
| $X$ | $X.p\_s$ | $X.s$ | $X \in \mathcal{K}_k$? |
| $\{e,p\}$ | 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 [STBPL01] 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

$$9')\qquad \mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s \text{ and } X.s \neq 1\}\ .$$

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

<u>$k = 3$:</u>

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 $\mathcal{K}_3$ 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 Figure 4.10.

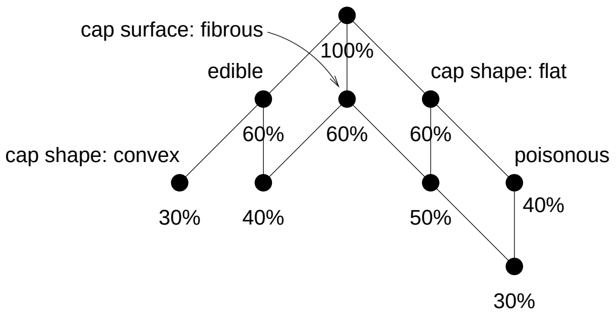

Figure 4.10: Iceberg concept lattice for the context in Figure 4.9 for minsupp = 30 %

### 4.4.6 Some Typical Applications

In Section 4.4.1, 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:

**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 [HSWW00], 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 clusters in more and more detail. In Figure 4.11, 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.

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:

**Discovery of functional dependencies.** One important task of logical database tuning is the discovery of minimal functional dependencies from database relations [HKPT99, LPL00]. 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 [KM95]. For this application, all 'min' in this section have to be replaced by 'max' (refer to Remark 2).

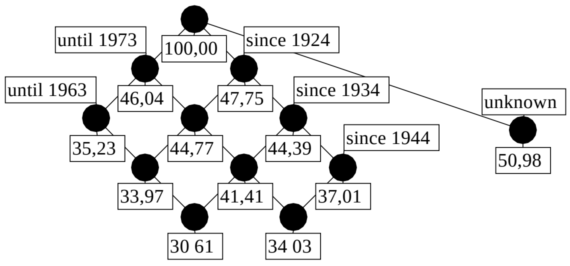

Figure 4.11: Iceberg concept lattice for customers clustered by their year of birth.

### 4.4.7 Complexity

There are several algorithms known for computing concept lattices: [MiS89], [GR91], [GM94], [NR99], [PBTL99a], [PBTL99b], and [PHM00]. The most efficient algorithm for practical applications to the best of our knowledge is Ganter's Next-Closure algorithm [GR91]; the algorithm with the best worst-case complexity is the In this section, we will compare TITANIC with Next Closure (Section 4.3) and an algorithm from Nourine and Raynaud [NR99], which substantiates in an efficient way the approach described in Section 2.3.1.

As shown in Section 4.3.4, the problem of computing concept lattices has exponential worst-case complexity. However, for practical purposes, it is interesting to examine the situation in more detail. In the sequel, we assume that $|M| \leq |G|$.

The Next Closure algorithm computes the concepts sequentially. In Section 4.3.5 is shown that the complexity for computing one concept is in $O(|G| \cdot |M|^2)$, so that the overall complexity could be stated as $O(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (|G| \cdot |M|^2))$. 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 $\left\lfloor \frac{|M|}{2} \right\rfloor$-sets are candidate sets. Then all these

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

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

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

## 4.5 Exercises

1. Give a proof of Lemma 10.
2. Give a proof of Theorem 11.
3. Let $M$ be a set. Show that the set $\mathfrak{C} := \{\mathcal{C} \subseteq \mathfrak{P}(M) \mid (C) \text{ is a closure system on } M\}$ is a closure system on $\mathfrak{P}(M)$.

## 4.6 Bibliographic Notes

———(to be written)———

There are several algorithms known for computing concept lattices: [MiS89], [GR91], [GM94], [YLBC96], [NR99], [PBTL99a], [PBTL99b], and [PHM00]. The most efficient algorithm for practical applications to the best of our knowledge is Ganter's Next-Closure algorithm [GR91]; the algorithm with the best worst-case complexity is the one from Nourine and Raynaud presented in [NR99]. The latter one substantiates in an efficient way the approach proposed by R. Wille [Wi82], which we presented in Section 2.3.1.

The way of computing concept lattices with TITANIC follows a data mining viewpoint by using a level-wise approach [AS94, MT97]. TITANIC was presented in [1], where also an experimental evaluation of its performance is discussed. Conceptual Clustering techniques were introduced first in [Mi80], see also [WMJ00]. FCA was considered as a framework for conceptual clustering from the early 1990ies on [StrW93, CR93, MG95].

FCA has been used as a formal framework for implication and association rules discovery and reduction [PBTL99a, STBPL01] and for improving the response times of algorithms for mining association rules [PBTL99b, PBTL99a]. The interaction of FCA and KDD in general has been discussed in [SWW98] and [HSWW00].