Fast Computation of Concept Lattices Using Data Mining Techniques

Gerd Stumme,[1] Rafik Taouil,[2] Yves Bastide,[2]
Nicolas Pasquier,[2] Lotfi Lakhal[2]

[1] Technische Universität Darmstadt, Fachbereich Mathematik,
Schloßgartenstr. 7, D–64289 Darmstadt, Germany;
stumme@mathematik.tu-darmstadt.de
[2] Laboratoire d'Informatique (LIMOS), Université Blaise Pascal,
Complexe Scientifique des Cézeaux, 24 Av. des Landais,
F–63177 Aubière Cedex, France;
{taouil,bastide,pasquier,lakhal}@libd2.univ-bpclermont.fr

Abstract

We present a new algorithm called TITANIC for computing concept lattices. It is based on data mining techniques for computing frequent itemsets. The algorithm is experimentally evaluated and compared with B. Ganter's Next-Closure algorithm.

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 80ies as a formalization of the concept of 'concept' [18], FCA has over the years grown to a powerful theory for data analysis, information retrieval, and knowledge discovery [14]. In Artificial Intelligence (AI), FCA is used as a knowledge representation mechanism. In database theory, FCA has been extensively used for class hierarchy design and management [12, 17, 19, 9]. 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 [16].

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 [4, 11, 15] and for improving the response times of algorithms for mining association rules [10, 11]. The interaction of FCA and KDD in general has been discussed in [13] and [5].

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 [9, 2, 4, 19]. We address the problem of computing concept lattices from a data mining viewpoint by using a level-wise approach [1, 8]; and provide a new, efficient algorithm called TITANIC.

In the next section, we present the theoretical foundation. It is turned into pseudo-code in Section 3. We conclude the paper with results of an experimental evaluation. Because of lack of space we will not provide proofs, and will not discuss the general use of FCA in AI and database theory.

2 Computing Concept Lattices by Using the Support Function

2.1 Formal Concept Analysis

In the first part of this section, we briefly recall the basic notions of Formal Concept Analysis. For a more extensive introduction into Formal Concept Analysis refer to [3].

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 items. *The inclusion* $(g, m) \in I$ *is read as "object* g *has attribute* m*". In this paper we assume that all sets are finite, especially* G *and* M*.*

For $A \subseteq G$*, we define* $A' := \{m \in M \mid \forall g \in A\colon (g, m) \in I\}$*; and for* $B \subseteq M$*, we define dually* $B' := \{g \in G \mid \forall m \in B\colon (g, m) \in I\}$*.* A formal concept *is a pair* (A, B) *with* $A \subseteq G$*,* $B \subseteq M$*,* $A' = B$ *and* $B' = A$*.* A *is called* extent *and* B *is called* intent *of the concept. The set of all concepts of a formal context* $\mathbb{K}$ *together with the partial order* $(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2$ *(which is equivalent to* $B_1 \supseteq B_2$*) is a complete lattice, called* concept lattice *of* $\mathbb{K}$*.*

	Jacobs	Plus	classic	mild	light	< 6 DM	< 8 DM	> 8 DM
Dallmayr Prodomo			×					×
Jacobs Krönung	×		×				×	
Jacobs Krönung Light	×				×		×	
Jacobs Krönung Free	×				×		×	
Jacobs Krönung Mild	×			×			×	
Jacobs Meisterröstung	×		×				×	
Tempelmann			×					×
Plus Schonkaffee		×			×		×	
Plus Naturmild		×		×		×	×	
Plus milde Sorte		×		×		×	×	
Plus Gold		×	×			×	×	
Idee Kaffee Classic			×					×
Kaffee Hag klassisch			×					×
Melitta Cafe Auslese			×				×	
Melitta Cafe Auslese Mild				×			×	
Kaisers Kaffee Auslese Mild				×			×	

Figure 1: Formal context about coffee brands sold in a supermarket.

Figure 1 shows a formal context which lists all coffee brands sold in a supermarket. Figure 2 shows the concept lattice of the context by a line diagram. In the *line diagram*, the name of an object g is always attached to the circle representing the

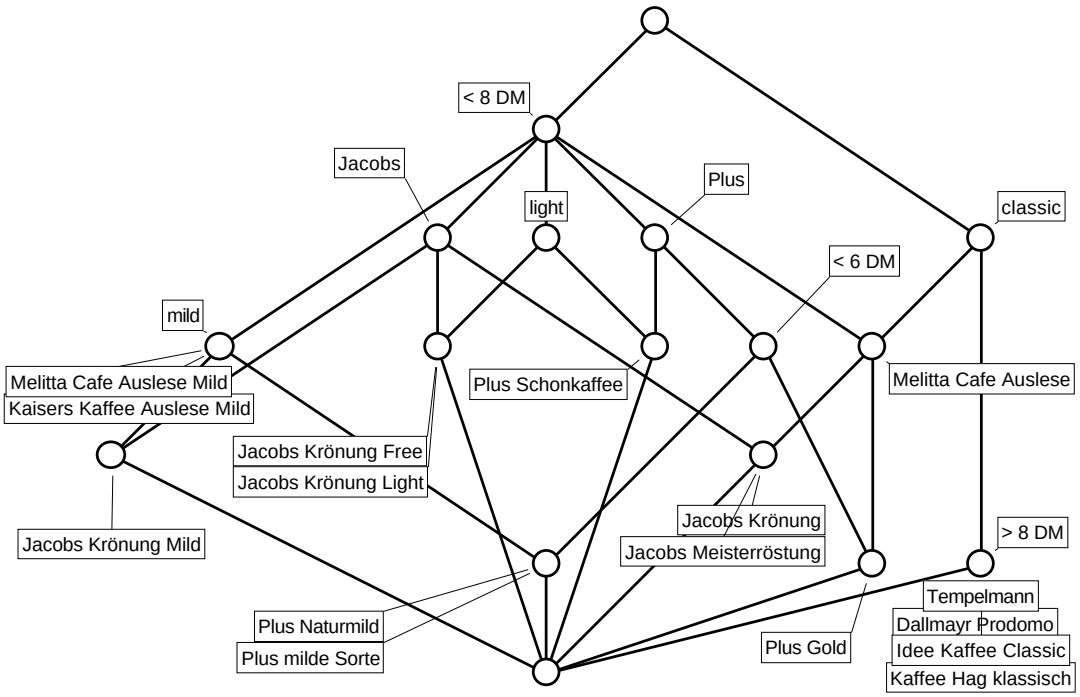

Figure 2: The concept lattice of the context in Figure 1

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 diagram, and the intent consists of all attributes attached to concepts above in the hierarchy. For example, the concept labeled by '< 6 DM' has {'Plus Naturmild', 'Plus milde Sorte', 'Plus Gold'} as extent, and {'< 6 DM', 'Plus' [the house brand of the supermarket], '< 8 DM'} as intent.

For $X, Y \subseteq M$, we say that the *implication* $X \to Y$ *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 with 100% confidence). For instance, the implication {Plus, classic} $\to$ {< 6 DM} holds in the coffee context. Implications can be read directly in the line diagram: the largest concept having both 'Plus' and 'classic' in its intent is below the concept labeled by '< 6 DM'. In [15] is shown how also the association rules with less than 100 % confidence can by visualized in the line diagram.

2.2 Support-Based Computation of the Closure System of all Concept Intents

In the following, we will use the composed function $\cdot'' \colon \mathfrak{P}(M) \to \mathfrak{P}(M)$ which is a closure operator on M (i. e., it is extensive, monotonous, and idempotent). 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 already determined by this closure system. Hence we restrict ourselves to the computation of the closure system of all concept intents in the sequel. The computation makes extensive use of the following *support* function:

Definition 2 *The* support *of* $X \subseteq M$ *is defined by* $\mathrm{supp}(X) := \frac{|X'|}{|G|}$.

In the case of X and Y with $X'' = Y''$, both sets have obviously the same support. On the other hand, comparable attribute sets with the same support also have the same closures:

Lemma 1 *Let* $X, Y \subseteq M$.
(i) $X'' = Y'' \Longrightarrow \mathrm{supp}(X) = \mathrm{supp}(Y)$
(ii) $X \subseteq Y \wedge \mathrm{supp}(X) = \mathrm{supp}(Y) \Longrightarrow X'' = Y''$

This lemma allows us to develop the algorithm in a more general setting:

Definition 3 *A* weight function *on* $\mathfrak{P}(M)$ *is a function* $s\colon \mathfrak{P}(M) \to P$ *from the powerset of* M *to a partially ordered set* $(P, \leq)$. *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) $X \subseteq Y \Longrightarrow s(X) \geq s(Y)$,[1] (ii) $h(X) = h(Y) \Longrightarrow s(X) = s(Y)$, (iii) $X \subseteq Y \wedge s(X) = s(Y) \Longrightarrow h(X) = h(Y)$.

Let h be a closure operator on a finite set M, and let s be a compatible weight function. The task is now to determine efficiently the closure system $\mathcal{H} := \{X \subseteq M \mid h(X) = X\}$ related to the closure operator h.

It is easy to check, that for a given formal context, the support function fulfills the conditions of Definition 3 for the closure operator $h(X) := X''$. Another problem where such a weight function can be used is the computation of the closure system induced by those functional dependencies which are valid for the actual data of a relational database (refer to [6]).

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.

[1] If $X \subseteq Y \Longrightarrow s(X) \leq s(Y)$ holds instead of (i) (as e. g. for functional dependencies), then all 'min' in the sequel (beside the one in Definition 6) have to be replaced by 'max'.

2.2.1 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 2 *Let $X \subseteq M$. Then*

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

Hence if we know the weights of all sets, then we can compute the closure operator ($\rightarrow$ Algorithm 3, steps 3–7).[2] 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 2.2.3 we discuss how the weights needed for those computations can be determined.

2.2.2 A level-wise approach for computing all closed sets

One can now compute the closure system $\mathcal{H}$ by applying Proposition 2 to all subsets X of M. But this is not efficient, since many closed sets will be determined several times.

Definition 4 *We define an equivalence relation θ on the powerset $\mathfrak{P}(M)$ of M by $(X, Y) \in \theta :\Longleftrightarrow h(X) = h(Y)$, for $X, Y \subseteq M$. The equivalence class of X is given by $[X] := \{Y \subseteq M \mid (X, Y) \in \theta\}$.*

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.

Definition 5 *A k-*set *is a subset X of M with $|X| = k$. For $\mathcal{X} \subseteq \mathfrak{P}(M)$, we define $\mathcal{X}_k := \{X \in \mathcal{X} \mid |X| = k\}$.*

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 6 *A set $X \subseteq M$ is a* key set *(or* minimal generator*) if $X \in \min[X]$. The set of all key sets is denoted by $\mathcal{K}$.*

Obviously we have $\mathcal{H} = \{h(X) \mid X \in \mathcal{K}\}$.

Proposition 3 *The set $\mathcal{K}$ is an order ideal of $(\mathfrak{P}(M), \subseteq)$ (i. e., $X \in \mathcal{K}$, $Y \subseteq X \Longrightarrow Y \in \mathcal{K}$).*

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

We will use a pruning strategy given in [1]. Originally this strategy was presented as a heuristic for determining all frequent sets only (which are, in our terminology, all sets with weights above a user-defined threshold). We show that this strategy can be applied to arbitrary order ideals of the powerset of M:

Definition 7 *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 $\mathcal{I}$.*

This definition is justified by the following lemma:

Lemma 4 *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 $\downarrow \mathcal{X}$ (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.*

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

Proposition 5 *Let $\mathcal{X} \subseteq \mathfrak{P}_{k-1}(M)$. Let $\widetilde{\mathcal{C}} := \{\{x_1, \ldots, x_k\} \mid i < j \Longrightarrow x_i < x_j, \{x_1, \ldots, x_{k-2}, x_{k-1}\}, \{x_1, \ldots, x_{k-2}, x_k\} \in \mathcal{X}\}$; and $\mathcal{C} := \{X \in \widetilde{\mathcal{C}} \mid \forall x \in X \colon X \setminus \{x\} \in \mathcal{X}\}$. Then $\mathcal{C} = \{X \in \mathfrak{P}_k(M) \mid X$ is candidate set for $\downarrow \mathcal{X}\}$.*

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

Proposition 6 *Let $X \subseteq M$.*
(*i*) *Let $m \in X$. Then $X \in [X \setminus \{m\}]$ if and only if $s(X) = s(X \setminus \{m\})$.*
(*ii*) *X is a key set if and only if $s(X) \neq \min_{m \in X}(s(X \setminus \{m\}))$.*

2.2.3 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 7 *If X is not a key set, then*

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

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

3 The Titanic Algorithm

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

Algorithm 1 Titanic

1) $\emptyset.s \leftarrow 1$;
2) $\mathcal{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) **forall** $m \in M$ **do** $\{m\}.p\_s \leftarrow 1$;
5) $\mathcal{C} \leftarrow \{\{m\} \mid m \in M\}$;
6) **loop begin**
7) Weigh($\mathcal{C}$);
8) $\mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s\}$;
9) **forall** $X \in \mathcal{K}_k$ **do** X.closure $\leftarrow$ Closure(X);
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** $\bigcup_{i=0}^{k-1}\{X.\text{closure} \mid X \in \mathcal{K}_i\}$.

Table 1: Notations used in Titanic

k	is the counter which indicates the current iteration. In the kth iteration, all key k-sets are determined.
$\mathcal{K}_k$	contains after the kth 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 8 to prune all non-key sets.

The algorithm starts with stating that the empty set is always a key set, and that its weight is — in the case of concept lattices — always equal to 1 (steps 1+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/Algorithm 2) which is (except step 5) a straight-forward implementation of Proposition 5. (The result of step 5 will be used in step 8 of Algorithm 1 for pruning the non-key sets.)

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). 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. 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.

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 $\{p_1, \ldots, p_k\}$ which are generated in step 1 are initialized by $\{p_1, \ldots, p_k\}.p\_s \leftarrow 1$.

1) $\mathcal{C} \leftarrow \{\{p_1, \ldots, p_k\} \mid i < j \Longrightarrow p_i < p_j,$
$\{p_1, \ldots, p_{k-2}, p_{k-1}\}, \{p_1, \ldots, p_{k-2}, p_k\} \in \mathcal{K}_{k-1}\}$;
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** $\mathcal{C}$.

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.

In step 8 of Algorithm 1, all candidate k-sets which are not key sets are pruned according to Proposition 6 (ii). For the remaining sets (which are now known to be key sets) their closures are computed (step 9). The CLOSURE function (Algorithm 3) is a straight-forward implementation of Propositions 3 and 7 (beside an additional optimization (step 2)). Algorithm 1 terminates, if there are no key k-sets left (step 11+14). Otherwise the next iteration begins (steps 10+12).

4 Experimental Evaluation and Conclusion

Several algorithms have been proposed for computing concept lattices. The most efficient at the best of our knowledge is Ganter's Next-Closure algorithm [2]. For our experimental evaluation, a version of the Titanic algorithm was implemented in C++ together with a rewriting of Ch. Lindig's C version of Next-Closure [7]. The comparisons took place on a Pentium III running at 600 MHz, with 512 MB of main memory, and were performed on the MUSHROOM (8,416 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 listed in Table 2 and visualized in Figure 3. They show that on the relatively strongly corelated MUSHROOMS database, Next-Closure is faster for few attributes, but takes twice the time of TITANIC for the whole dataset. On the weakly corelated INTERNET database, the difference is much larger. This stems from the fact that the development of TITANIC was inspired by the Apriori algorithm which is known to perform well on weakly corelated data.

Table 2: Database characteristics and evaluation results

Database	# of objects	# of attr.	# of concepts	Computation time (sec.) Next-Closure	Titanic
Mushrooms	2,500	79	5,394	31.13	48.19
	5,000	79	9,064	108.38	75.59
	8,416	80	32,086	527.74	200.73
Internet	1,000	141	15,107	16.49	4.33
	2,000	141	31,719	66.32	7.31
	5,000	141	73,026	381.95	14.31
	7,500	141	100,706	803.17	19.13
	10,000	141	124,574	1431.86	23.46

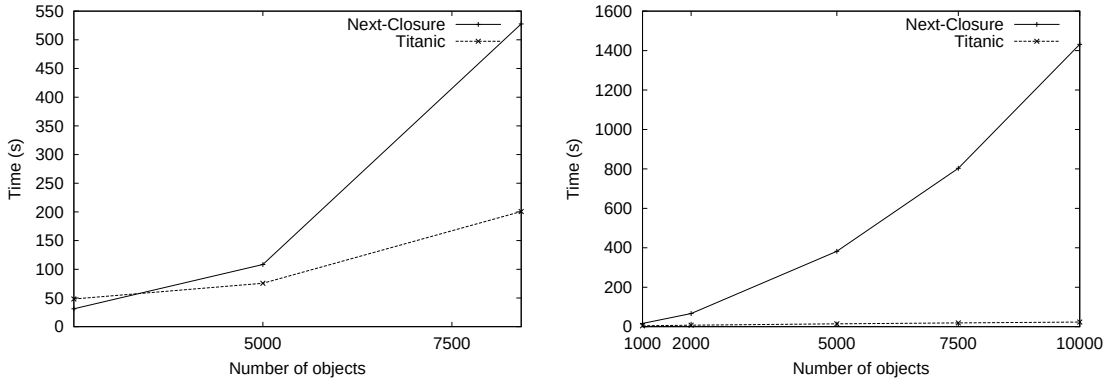

Figure 3: Comparison of TITANIC and Next-Closure on the MUSHROOMS (left) and INTERNET databases (right)

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 show that under normal conditions (and if handled with care) a strong and waterproof algorithm may improve the exploration of unknown regions of knowledge.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. VLDB Conf.*, 1994, 478–499 (Expanded version in IBM Report RJ9839)

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

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

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

[5] J. Hereth, G. Stumme, U. Wille, R. Wille: Conceptual Knowledge Discovery and Data Analysis. In: B. Ganter, G. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Structures*. Proc. ICCS 2000. Springer, Heidelberg 2000 (to appear)

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

[7] 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)

[8] H. Mannila, H. Toivonen: Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery* **1**(3): 241-258 (1997)

[9] M. Missikoff, M. Scholl: An algorithm for insertion into a lattice: application to type classification. *Proc. 3rd Intl. Conf. FODO 1989*. LNCS **367**, Springer, Heidelberg 1989, 64–82

[10] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Discovering frequent closed itemsets for association rules. *Proc. ICDT Conf.*, 1999, 398–416

[11] 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

[12] Ingo Schmitt, Gunter Saake: Merging inheritance hierarchies for database integration. *Proc. 3rd IFCIS Intl. Conf. on Cooperative Information Systems*, New York City, Nework, USA, August 20-22, 1998, 122–131

[13] G. Stumme, R. Wille, U. Wille: Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods. In: J. M. Żytkow, M. Quafofou (eds.): *Principles of Data Mining and Knowledge Discovery*. Proc. 2nd European Symposium on PKDD '98, LNAI **1510**, Springer, Heidelberg 1998, 450–458

[14] G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung – Methoden und Anwendungen*. Springer, Heidelberg 2000

[15] R. Taouil, N. Pasquier, Y. Bastide, G. Stumme, L. Lakhal: Mining bases for association rules based on formal concept analysis. *Journal on Knowledge and Information Systems* (submitted)

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

[17] K. Waiyamai, R. Taouil, L. Lakhal: Towards an object database approach for managing concept lattices. *Proc. 16th Intl. Conf. on Conceptual Modeling*, LNCS **1331**, Springer, Heidelberg 1997, 299-312

[18] R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.). *Ordered sets*. Reidel, Dordrecht–Boston 1982, 445–470

[19] A. Yahia, L. Lakhal, J. P. Bordat, R. Cicchetti: iO2: An algorithmic method for building inheritance graphs in object database design. *Proc. 15th Intl. Conf. on Conceptual Modeling*. LNCS **1157**, Springer, Heidelberg 1996, 422–437