Ann Math Artif Intell (2014) 70:5–24
DOI 10.1007/s10472-013-9353-y

Optimizations in computing the Duquenne–Guigues basis of implications

Konstantin Bazhanov · Sergei Obiedkov

Published online: 9 May 2013
© Springer Science+Business Media Dordrecht 2013

Abstract In this paper, we consider algorithms involved in the computation of the Duquenne–Guigues basis of implications. The most widely used algorithm for constructing the basis is Ganter's Next Closure, designed for generating closed sets of an arbitrary closure system. We show that, for the purpose of generating the basis, the algorithm can be optimized. We compare the performance of the original algorithm and its optimized version in a series of experiments using artificially generated and real-life datasets. An important computationally expensive subroutine of the algorithm generates the closure of an attribute set with respect to a set of implications. We compare the performance of three algorithms for this task on their own, as well as in conjunction with each of the two algorithms for generating the basis. We also discuss other approaches to constructing the Duquenne–Guigues basis.

Keywords Implications · Horn formulae · Duquenne–Guigues basis · Formal Concept Analysis · Algorithms · LinClosure

Mathematics Subject Classifications (2010) 06A15 · 68T99 · 68W05 · 68W40

1 Introduction

Implications are among the most important tools of formal concept analysis (FCA) [15]. They are closely related to Horn formulae: an implication can be represented

K. Bazhanov · S. Obiedkov (✉)
National Research University Higher School of Economics, Moscow, Russia
e-mail: sergei.obj@gmail.com

K. Bazhanov
e-mail: kostyabazhanov@mail.ru

as a conjunction of definite Horn clauses with the same body. The set of all attribute implications valid in a formal context defines a closure operator mapping attribute sets to concept intents of the context (this mapping is surjective). The following two algorithmic problems arise with respect to implications:

1. Given a set $\mathcal{L}$ of implications and an attribute set A, compute the closure $\mathcal{L}(A)$.
2. Given a formal context $\mathbb{K}$, compute a set of implications equivalent to the set of all implications valid in $\mathbb{K}$, i.e., the *cover* of valid implications.

The first of these problems has received considerable attention in the database literature in application to functional dependencies [22]. Although functional dependencies are interpreted differently than implications, the two are in many ways similar: in particular, they share the notion of semantic consequence and the syntactic inference mechanism (Armstrong rules [4]). A linear-time algorithm, LinClosure, has been proposed for computing the closure of a set with respect to a set of functional dependencies (or implications) [7], i.e., for solving the first of the two problems stated above. However, the asymptotic complexity estimates may not always be good indicators for relative performance of algorithms in practical situations. In Section 3, we compare LinClosure with two other algorithms—a "naïve" algorithm, Closure [22], and the algorithm proposed in [31]—both of which are non-linear. We analyze their performance in several particular cases and compare them experimentally on several datasets.

For the second problem, an obvious choice of the cover is the Duquenne–Guigues, or canonical, basis of implications, which is the smallest set equivalent to the set of valid implications [16]. Unlike for the other frequently occurring FCA algorithmic task, the computation of all formal concepts of a formal context [20], only few algorithms have been proposed for the calculation of the canonical basis. The most widely-used algorithm was proposed by Ganter in [13]. Another, attribute-incremental, algorithm for the same problem was described in [24]. It is claimed to be much faster than Ganter's algorithm for most practical situations. The Concept Explorer software system [32] uses this algorithm to generate the Duquenne–Guigues basis of a formal context. However, we do not discuss it here, for we choose to concentrate on the computation of implications in the lectic order (see Section 4). The lectic order is important in the interactive knowledge-acquisition procedure of attribute exploration [14], where implications are output one by one and the user is requested to confirm or reject (by providing a counterexample) each implication (see [5, 25, 26] for some applications of attribute exploration).

Ganter's algorithm repeatedly computes the closure of an attribute set with respect to a set of implications; therefore, it relies heavily on a subprocedure implementing a solution to the first problem. In Section 4, we propose optimizations to Ganter's algorithm and experimentally compare the original and optimized versions in conjunction with each of the three algorithms for solving the first problem. In Section 5, we briefly discuss other approaches to generating the implication basis, including algorithms originating within formal concept analysis and those developed by the artificial intelligence community in a rather different setting, namely, that of learning Horn theories with queries [2] and approximating knowledge bases by Horn theories [17]. A systematic comparison with these other approaches is left for further work.

2 The Duquenne–Guigues basis of implications

Before proceeding, we quickly recall the definition of the Duquenne–Guigues basis and related notions.

Given a *(formal) context* $\mathbb{K} = (G, M, I)$, where G is called a set of *objects*, M is called a set of *attributes*, and the binary relation $I \subseteq G \times M$ specifies which objects have which attributes, the derivation operators $(\cdot)^I$ are defined for $A \subseteq G$ and $B \subseteq M$ as follows:

$$A' = \{m \in M \mid \forall g \in A : gIm\} \qquad B' = \{g \in G \mid \forall m \in B : gIm\}$$

In words, A' is the set of attributes common to all objects of A and B' is the set of objects sharing all attributes of B. The double application of $(\cdot)'$ is a closure operator, i.e., $(\cdot)''$ is extensive, idempotent, and monotonous. Therefore, sets A'' and B'' are said to be *closed*. Closed object sets are called *concept extents* and closed attribute sets are called *concept intents* of the formal context $\mathbb{K}$.

In discussing the algorithms later in the paper, we assume that the sets G and M are finite.

An *implication* over M is an expression $A \to B$, where $A, B \subseteq M$ are attribute subsets. It *holds* or is *valid* in the context if $A' \subseteq B'$, i.e., every object of the context that has all attributes from A also has all attributes from B. The size of A' is sometimes called the *support* of the valid implication $A \to B$.

An attribute subset $X \subseteq M$ *respects* (or is a *model* of) an implication $A \to B$ if $A \nsubseteq X$ or $B \subseteq X$. Obviously, an implication holds in a context (G, M, I) if and only if $\{g\}'$ respects the implication for all $g \in G$.

A set $\mathcal{L}$ of implications over M defines the closure operator $X \mapsto \mathcal{L}(X)$ that maps $X \subseteq M$ to the smallest set respecting all the implications in $\mathcal{L}$:[1]

$$\mathcal{L}(X) = \bigcap\{Y \mid X \subseteq Y \subseteq M, \forall(A \to B) \in \mathcal{L} : A \nsubseteq Y \text{ or } B \subseteq Y\}.$$

We discuss algorithms for computing $\mathcal{L}(X)$ in Section 3. Note that, if $\mathcal{L}$ is the set of all valid implications of a formal context, then $\mathcal{L}(X) = X''$ for all $X \subseteq M$.

Two implication sets over M are *equivalent* if they are respected by exactly the same subsets of M. Equivalent implication sets define the same closure operator. A *minimum cover* of an implication set $\mathcal{L}$ is a set of minimal size among all implication sets equivalent to $\mathcal{L}$. One particular minimum cover described in [16] is defined using the notion of a pseudo-closed set, which we introduce next.

A set $P \subseteq M$ is called *pseudo-closed* (with respect to a closure operator $(\cdot)''$) if $P \neq P''$ and $Q'' \subset P$ for every pseudo-closed $Q \subset P$.

In particular, all minimal non-closed sets are pseudo-closed. A pseudo-closed attribute set of a formal context is also called a *pseudo-intent*.

The *Duquenne–Guigues* or *canonical basis* of implications (with respect to a closure operator $(\cdot)''$) is the set of all implications of the form $P \to P''$, where P is pseudo-closed. This set of implications is of minimal size among those defining the closure operator $(\cdot)''$. If $(\cdot)''$ is the closure operator associated with a formal

[1]We deliberately use the same letter $\mathcal{L}$ for an implication set and the closure operator it defines.

context, the Duquenne–Guigues basis is a minimum cover of valid implications of this context. The computation of the Duquenne–Guigues basis of a formal context is hard—at least as hard as the well-known hypergraph transversal problem (see [12, 18] for a reduction), for which no output-polynomial algorithm is known. We discuss algorithms for computing the basis in Section 4.

Example 1 A formal context can be visualized by means of a cross-table, as shown below:

	a	b	c	d	e
1	×	×	×	×	×
2				×	×
3	×		×		
4	×				
5	×			×	

In this context, there are five objects (denoted by numbers 1, 2, 3, 4, and 5) and five attributes (denoted by a, b, c, d, and e). If an object has an attribute, the corresponding cell in the table contains a cross; otherwise, it is empty. The canonical basis of this context consists of five implications:

$$\begin{aligned}
\{e\} &\quad\to\quad \{d, e\}\\
\{c\} &\quad\to\quad \{a, c\}\\
\{b\} &\quad\to\quad \{a, b, c, d, e\}\\
\{a, d, e\} &\quad\to\quad \{a, b, c, d, e\}\\
\{a, c, d\} &\quad\to\quad \{a, b, c, d, e\}
\end{aligned}$$

Taking into account that the empty attribute set is closed in this context, it is easy to see that the three non-closed single-element sets $\{b\}$, $\{c\}$, and $\{e\}$ satisfy the definition of a pseudo-intent. The pseudo-intent $\{a, c, d\}$ properly contains the pseudo-intent $\{c\}$, but it also contains its closure $\{a, c\}$. In its turn, $\{a, c, d\}$ is a pseudo-closed proper subset of $\{a, b, c, d\}$, which however does not contain the closure of $\{a, c, d\}$; therefore, $\{a, b, c, d\}$ is not pseudo-closed. In general, it may be hard to decide if a set is pseudo-closed in a given formal context: although the problem of recognizing pseudo-intents is in coNP [21], it is complete for this class [6].

3 Computing the closure of an attribute set

In this section, we compare the performance of algorithms computing the closure of an attribute set X with respect to a set $\mathcal{L}$ of implications. Algorithm 1 [22] checks every implication $A \to B \in \mathcal{L}$ and enlarges X with attributes from B if $A \subseteq X$. The algorithm terminates when a fixed point is reached, that is, when the set X cannot be enlarged any further (which always happens at some moment, since both $\mathcal{L}$ and M are assumed finite).

The algorithm is obviously quadratic in the number of implications in $\mathcal{L}$ in the worst case. The worst case happens when exactly one implication is applied at each

Algorithm 1 Closure(X, $\mathcal{L}$)

Input: An attribute set $X \subseteq M$ and a set $\mathcal{L}$ of implications over M.
Output: The closure of X w.r.t. implications in $\mathcal{L}$.

```
repeat
    $stable$ := true
    for all $A \to B \in \mathcal{L}$ do
        if $A \subseteq X$ then
            $X := X \cup B$
            $stable$ := false
            $\mathcal{L} := \mathcal{L} \setminus \{A \to B\}$
until $stable$
return $X$
```

iteration (but the last one) of the **repeat** loop, resulting in $|\mathcal{L}|(|\mathcal{L}| + 1)/2$ iterations of the **for all** loop, each requiring $O(|M|)$ time.

Example 2 A simple example of the worst case is when $X = \{1\}$ and the implications in $\mathcal{L} = \{\{i\} \to \{i + 1\} \mid i \in \mathbb{N}, 0 < i < n\}$ for some n are arranged in the descending order of their one-element premises.

In [7], a linear-time algorithm, LinClosure, is proposed for the same problem. Algorithm 2 is identical to the version of LinClosure from [22] except for one modification designed to allow implications with empty premises in $\mathcal{L}$. LinClosure associates a counter with each implication initializing it with the size of the implication premise. Also, each attribute is linked to a list of implications that have it in their premises. The algorithm then checks every attribute m of X (the set whose closure must be computed) and decrements the counters for all implications linked to m. If the counter of some implication $A \to B$ reaches zero, attributes from B are added to X. Afterwards, they are used to decrement counters along with the original attributes of X. When all attributes in X have been checked in this way, the algorithm stops with X containing the closure of the input attribute set.

It can be shown that the algorithm is linear in the length of the input assuming that each attribute in the premise or conclusion of any implication in $\mathcal{L}$ requires a constant amount of memory [22].

Example 3 The worst case for LinClosure occurs, for instance, when $X \subset \mathbb{N}$, $M = X \cup \{1, 2, \ldots, n\}$ for some n such that $X \cap \{1, 2, \ldots, n\} = \varnothing$ and $\mathcal{L}$ consists of implications of the form

$$X \cup \{i \mid 0 < i < k\} \to \{k\}$$

for all k such that $1 \leq k \leq n$. During each of the first $|X|$ iterations of the **for all** loop (nested inside the **while** loop), the counters of all implications will have to be updated with only the last iteration adding one attribute to X using the implication $X \to \{1\}$. At each of the subsequent $n - 1$ iterations, the counter for every so far "unused" implication will be updated and one attribute will be added to X. The next, $(|X| + n)$th, iteration will terminate the algorithm.

Algorithm 2 LinClosure(X, $\mathcal{L}$)

Input: An attribute set $X \subseteq M$ and a set $\mathcal{L}$ of implications over M.
Output: The closure of X w.r.t. implications in $\mathcal{L}$.

```
for all $A \to B \in \mathcal{L}$ do
    $count[A \to B] := |A|$
    if $|A| = 0$ then
        $X := X \cup B$
    for all $a \in A$ do
        add $A \to B$ to $list[a]$
$update := X$

while $update \neq \varnothing$ do
    choose $m \in update$
    $update := update \setminus \{m\}$
    for all $A \to B \in list[m]$ do
        $count[A \to B] = count[A \to B] - 1$
        if $count[A \to B] = 0$ then
            $add := B \setminus X$
            $X := X \cup add$
            $update := update \cup add$

return $X$
```

Note that, if the implications in $\mathcal{L}$ are arranged in the reversed subset-inclusion order of their premises, this example will present the worst case for Algorithm 1 requiring n iterations of the main loop. However, if the implications are arranged in the subset-inclusion order of their premises, one iteration will be sufficient.

Inspired by the mechanism used in LinClosure to obtain linear asymptotic complexity, but somewhat disappointed by the poor performance of the algorithm relative to Closure, which was revealed in his experiments, Marcel Wild proposed a new algorithm in [31]. We present this algorithm (in a slightly more compact form) as Algorithm 3. The idea is to maintain implication lists similar to those used in LinClosure, but get rid of the counters. Instead, at each step, the algorithm combines the implications in the lists associated with attributes not occurring in X and "fires" the remaining implications (i.e., uses them to enlarge X). When there is no implication to fire, the algorithm terminates with X containing the desired result.

Wild claims that his algorithm is faster than both LinClosure and Closure, even though it has the same asymptotic complexity as the latter. The worst case for Algorithm 3 is when $\mathcal{L} \setminus \mathcal{L}_1$ contains exactly one implication $A \to B$ and $B \setminus X$ contains exactly one attribute at each iteration of the **repeat... until** loop. Example 2 presents the worst case for Algorithm 3, but, unlike for Closure, the order of implications in $\mathcal{L}$ is irrelevant. The worst case for LinClosure from Example 3 is also the worst case for Algorithm 3, but it deals with it, perhaps, in a more efficient way using n iterations of the main loop compared to $n + |X|$ iterations of the main loop in LinClosure.

Algorithm 3 WILD'S CLOSURE(X, $\mathcal{L}$)

Input: An attribute set $X \subseteq M$ and a set $\mathcal{L}$ of implications over M.
Output: The closure of X w.r.t. implications in $\mathcal{L}$.

```
for all $m \in M$ do
    for all $A \to B \in \mathcal{L}$ do
        if $m \in A$ then
            add $A \to B$ to $list[m]$

repeat
    $stable := true$
    $\mathcal{L}_1 := \bigcup_{m \in M \setminus X} list[m]$
    for all $A \to B \in \mathcal{L} \setminus \mathcal{L}_1$ do
        $X := X \cup B$
        $stable :=$ false
    $\mathcal{L} := \mathcal{L}_1$
until $stable$

return $X$
```

3.1 Experimental comparison

We implemented the algorithms in C++ using Microsoft Visual Studio 2010. The implementation is available at https://github.com/kostyabazhanov/FCAImplications/. For the implementation of attribute sets, as well as sets of implications in Algorithm 3, we used dynamic bit sets from the Boost library [11]. All the tests described in the following sections were carried out on an Intel Core i5 2.67 GHz computer with 4 Gb of memory running under Windows 7 Home Premium x64.

Figure 1 shows the performance of the three algorithms on Example 2. Algorithm 2 is the fastest algorithm in this case: for a given n, it needs n iterations of the outer loop—the same as the other two algorithms, but the inner loop of Algorithm 2 checks exactly one implication at each iteration, whereas the inner loop of Algorithm 1 checks $n - i$ implications at the ith iteration. Although the inner loop

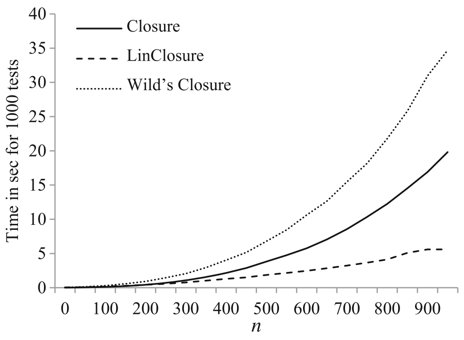


Fig. 1 Performance of Algorithms 1–3 on Example 2

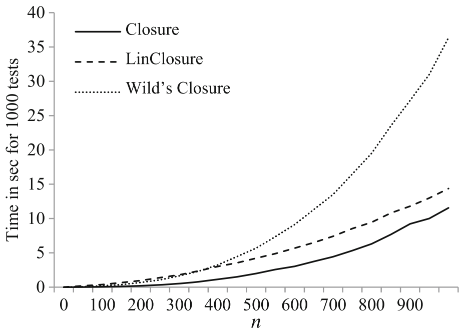


Fig. 2 Performance of Algorithms 1–3 on Example 3 with implications in $\mathcal{L}$ arranged in the reverse subset-inclusion order of their premises and $|X| = 50$

of Algorithm 3 checks only one implication at the ith iteration, it has to compute the union of $n - i$ lists in addition.

Figure 2 shows the performance of the algorithms on Example 3. Here, the behavior of Algorithm 2 is similar to that of Algorithm 1, but Algorithm 2 takes more time due to the complicated initialization step.

Interestingly, Algorithm 1 works almost twice as fast on Example 3 as it does on Example 2. This may seem surprising, since it is easy to see that the algorithm performs essentially the same computations in both cases, the difference being that the implications of Example 2 have single-element premises. However, this turns out to be a source of inefficiency: at each iteration of the main loop, all implications but the last fail to fire, but, for each of them, the algorithm checks if their premises are included in the set X. Generally, when $A \not\subseteq X$, this can be established easier if A is large, for, in this case, A is likely to contain more elements outside X. This effect is reinforced by the implementation of sets as bit strings: roughly speaking, to verify that $\{i\} \not\subseteq \{1\}$, it is necessary to check all bits up to i, whereas $\{i \mid 0 < i < k\} \not\subseteq \{k + 1\}$ can be established by checking only one bit (assuming that bits are checked from left to right). Alternative data structures for set implementation might have less dramatic consequences for performance in this setting. On the other hand, the example shows that performance may be affected by issues not so obviously related to the structure of the algorithm, thus, suggesting additional paths to obtain an optimal behavior (e.g., by rearranging attributes or otherwise preprocessing the input data).

We have experimented with computing closures using the canonical bases of formal contexts as input implication sets. Table 1 shows the results for randomly generated contexts. The first two columns indicate the size of the attribute set and the number of implications, respectively. The remaining three columns record the time (in seconds) for computing the closures of 1,000 randomly generated subsets of M by each of the three algorithms. Table 2 presents similar results for datasets taken from the UCI repository [9] and, if necessary, transformed into formal contexts using FCA scaling [15].[2] The contexts are described in Table 3, where the last four columns correspond to the number of objects, number of attributes, number of intents, and number of pseudo-intents (i.e., the size of the canonical basis) of the context named in the first column.

[2]The breast cancer domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia (now, Slovenia). Thanks go to M. Zwitter and M. Soklic for providing the data.

Table 1 Performance of Algorithms 1–3 on randomly generated tests (time in seconds per 1,000 closures)

		Algorithm		
$\|M\|$	$\|\mathcal{L}\|$	1	2	3
30	557	0.0051	0.2593	0.0590
50	1,115	0.0118	0.5926	0.1502
100	380	0.0055	0.2887	0.0900
100	546	0.0086	0.4229	0.1350
100	2,269	0.0334	1.5742	0.5023
100	3,893	0.0562	2.6186	0.8380
100	7,994	0.1134	5.3768	1.7152
100	8,136	0.1159	5.6611	1.8412

Table 2 Performance on the canonical bases of contexts from Table 3 (time in seconds per 1,000 closures)

	Algorithm		
Context	1	2	3
Zoo	0.0036	0.0905	0.0182
Postoperative patient	0.0054	0.2980	0.0722
Congressional voting	0.0075	0.1505	0.0883
SPECT	0.0251	0.9848	0.2570
Breast cancer	0.0361	1.7912	0.5028
Solar flare	0.0370	2.1165	0.6317
Wisconsin breast cancer	0.1368	8.4984	2.4730

Table 3 Contexts obtained from UCI datasets

Context	$\|G\|$	$\|M\|$	# intents	# pseudo-intents
Zoo	101	28	379	141
Postoperative patient	90	26	2,378	619
Congressional voting	435	18	10,644	849
SPECT	267	23	21,550	2,169
Breast cancer	286	43	9,918	3,354
Solar flare	1,389	49	28,742	3,382
Wisconsin breast cancer	699	91	9,824	10,666

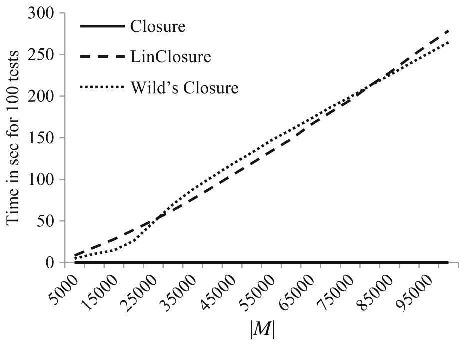


Fig. 3 Performance of Algorithms 1–3 when computing $\mathcal{L}(X)$ a hundred times, where $X \subseteq M$, $|X| = 100$, $\mathcal{L}$ is a randomly generated set of 1,000 implications over M, and $|M|$ runs from 5,000 to 100,000

In these experiments, Algorithm 1 was the fastest and Algorithm 2 was the slowest, even though it has the best asymptotic complexity. This can be partly explained by the large overhead of the initialization step (setting up counters and implication lists). Therefore, these results can be used as a reference only when the task is to compute one closure for a given set of implications. When a large number of closures must be computed with respect to the same set of implications, Algorithms 2 and 3 may be more appropriate.

Figure 3 shows the time each algorithm needs for computing the closure of a randomly generated subset of M if we fix the number of implications (here, $|\mathcal{L}| = 1{,}000$), but allow $|M|$ to grow from 5,000 to 100,000. One can see that the time required by Algorithms 2 and 3 grows linearly up to almost 300 seconds (per 100 closures), while the time needed by Algorithm 1 never gets too far from zero. The reason is that Algorithm 1 is quadratic in the number of implications, which is constant in this experiment. Of course, the growth of $|M|$ does slow down Algorithm 1, since it needs more time for each set-theoretic operation, but this is nothing compared to the increased overhead for maintaining the lists in Algorithms 2 and 3. In this scenario (many attributes and few implications), the simple Algorithm 1 is a clear winner.

4 Computing the basis in the lectic order

The best-known algorithm for computing the Duquenne–Guigues basis was developed by Ganter in [13]. The algorithm is based on the fact that intents and pseudo-intents of a context taken together form a closure system. This makes it possible to iteratively generate all intents and pseudo-intents using Next Closure (see Algorithm 4), a generic algorithm for enumerating closed sets of an arbitrary closure operator (also proposed in [13]). For every generated pseudo-intent P, an implication $P \to P''$ is added to the basis. The intents, which are also generated, are simply discarded.

Algorithm 4 Next Closure(A, M, $\mathcal{L}(\cdot)$)

Input: A closure operator $X \mapsto \mathcal{L}(X)$ on M and a subset $A \subseteq M$.
Output: The lectically next closed set after A.

```
for all $m \in M$ in reverse order do
    if $m \in A$ then
        $A := A \setminus \{m\}$
    else
        $B := \mathcal{L}(A \cup \{m\})$
        if $B \setminus A$ contains no element $< m$ then
            return $B$
return $\perp$
```

Next Closure takes a closed set as input and outputs the next closed set according to a particular *lectic* order, which is a linear extension of the subset-inclusion order. Assuming a linear order $<$ on attributes in M, we say that a set $A \subseteq M$ is *lectically smaller* than a set $B \subseteq M$ if

$$\exists b \in B \setminus A \; \forall a \in A (a < b \Rightarrow a \in B).$$

In other words, the lectically largest among two sets is the one containing the smallest element in which they differ.

Example 4 Let $M = \{a < b < c < d < e < f\}$, $A = \{a, c, e\}$ and $B = \{a, b, f\}$. Then, A is lectically smaller than B, since the first attribute in which they differ, b, is in B. Note that if we represent sets by bit strings with smaller attributes corresponding to higher-order bits (in our example, $A = 101010$ and $B = 110001$), the lectic order will match the usual less-than order on binary numbers.

To be able to use Next Closure for iterating over intents and pseudo-intents, we need access to the corresponding closure operator that maps an attribute subset to the smallest intent or pseudo-intent that includes it. This operator, which we denote by $^\bullet$, is defined via the Duquenne–Guigues basis $\mathcal{L}$ as follows. For a subset $A \subseteq M$, put

$$A^+ = A \cup \bigcup \{P'' \mid P \to P'' \in \mathcal{L}, P \subsetneq A\}.$$

Then, $A^\bullet = A^{++\cdots+}$, where $A^{\bullet+} = A^\bullet$; i.e., $^\bullet$ is the transitive closure of $^+$.

The problem is that $\mathcal{L}$ is not available when we start; in fact, this is precisely what we want to generate. Fortunately, for computing a pseudo-closed set A, it is sufficient to know only implications with premises that are proper subsets of A. Generating pseudo-closed sets in the lectic order, which is compatible with the subset-inclusion order, we ensure that, at each step, we have at hand the required part of the basis. Therefore, we can use any of the three algorithms from Section 3 to compute $A^\bullet$ (provided that the implication $A^\bullet \to A''$ has not been added to $\mathcal{L}$ yet). Algorithm 5 uses Next Closure to generate the canonical basis. It passes Next Closure the closure operator $\mathcal{L}(\cdot)$ associated with the part of the basis computed so far; any of the Algorithms 1–3 can be used to compute the closure, $\mathcal{L}(A \cup \{m\})$, with respect to the set $\mathcal{L}$ of implications.

Algorithm 5 Canonical Basis(M, $''$)

Input: A closure operator $X \mapsto X''$ on M, e.g., given by a formal context (G, M, I).
Output: The canonical basis for the closure operator.

```
$A := \varnothing$
$\mathcal{L} := \varnothing$
while $A \neq M$ do
    if $A \neq A''$ then
        $\mathcal{L} := \mathcal{L} \cup \{A \to A''\}$
    $A :=$ Next Closure($A, M, \mathcal{L}(\cdot)$)
return $\mathcal{L}$
```

After Next Closure computes $A^\bullet$, the implication $A^\bullet \to A''$ is added to the basis provided $A^\bullet \neq A''$. Algorithm 5 will then pass $A^\bullet$ as the input to Next Closure, but there is some room for optimization here. Let i be the maximal element of A and j be the minimal element of $A'' \setminus A$. Consider the following two cases:

$j < i$ As long as $m > i$, the set $\mathcal{L}(A^\bullet \cup \{m\})$ will be rejected by Next Closure, since it will contain j. Hence, it makes sense to skip all $m > i$ and continue as if

$A^\bullet$ had been rejected by NEXT CLOSURE. This optimization has already been proposed in [24].

$i < j$ From Proposition 1 below, it follows that, in this case, the lectically next intent or pseudo-intent after $A^\bullet$ is A''. Hence, we can skip all sets between $A^\bullet$ and A'' and proceed directly with A'' instead of $A^\bullet$ at the next step.

Proposition 1 *If P is pseudo-closed and no element of $P'' \setminus P$ is smaller than any element of P, then P is the lectically largest pseudo-closed set with closure P''.*

Proof If Q is a lectically larger pseudo-closed set with $Q'' = P''$, it must contain an element $q \notin P$ that is smaller than some element in P; but then q would also be contained in $Q'' = P''$ and in $P'' \setminus P$. □

Algorithm 6 takes these considerations into account.

Algorithm 6 CANONICAL BASIS(M, $''$), an optimized version

Input: A closure operator $X \mapsto X''$ on M, e.g., given by a formal context (G, M, I).
Output: The canonical basis for the closure operator.

```
$A := \varnothing''$
if $A = \varnothing$ then
    $\mathcal{L} := \varnothing$
else
    $\mathcal{L} := \{\varnothing \to A\}$
$i :=$ the largest element of $M$

while $A \neq M$ do

    for all $j \leq i \in M$ in reverse order do
        if $j \in A$ then
            $A := A \setminus \{j\}$
        else
            $B := \mathcal{L}(A \cup \{j\})$
            if $B \setminus A$ contains no element $< j$ then
                $A := B$
                $i := j$
                exit for

    if $A \neq A''$ then
        $\mathcal{L} := \mathcal{L} \cup \{A \to A''\}$
    if $A'' \setminus A$ contains no element $< i$ then
        $A := A''$
        $i :=$ the largest element of $M$
    else
        $A := \{m \in A \mid m \leq i\}$

return $\mathcal{L}$
```

Example 5 To get an idea of a speed-up that can be obtained by Algorithm 6 as compared to Algorithm 5, consider what happens at the moment each algorithm computes the implication

$$\{c\} \to \{a, c\}$$

when applied to the context from Example 1. Algorithm 5 will then request from Next Closure the lectically next intent or pseudo-intent after $\{c\}$. In its search for such a set, Next Closure will try $\{c, e\}$ and then $\{c, d\}$—and will fail on both occasions—before moving to $\{b\}$. On the contrary, Algorithm 6 is capable of sensing these failures by observing that $\{a, c\}$, the closure of $\{c\}$, which is computed anyway, contains attribute a preceding c. Therefore, Algorithm 6 moves directly to $\{b\}$. In general, for the ith attribute of M (in the role of c in this example), this saves us from considering $|M| - i$ hopeless candidates and from computing as many closures using one of Algorithms 1–3. Thus, after generating

$$\{b\} \to \{a, b, c, d, e\},$$

Algorithm 5 unsuccessfully attempts $\{b, e\}$, $\{b, d\}$, and $\{b, c\}$, while Algorithm 6 moves straight to $\{a\}$.

4.1 Experimental comparison

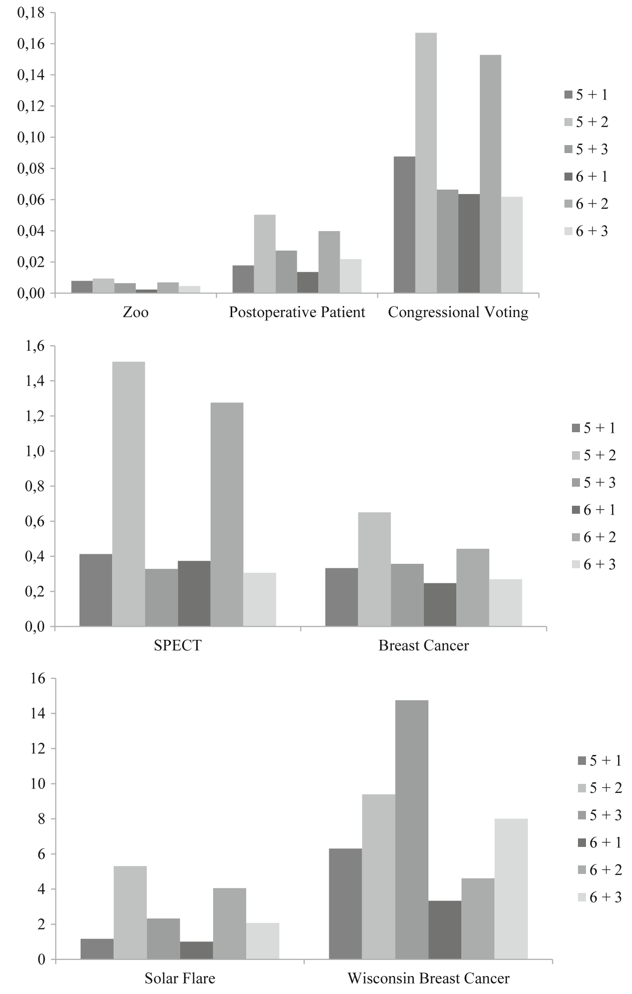
We used Algorithms 5 and 6 for constructing the canonical bases of the contexts involved in testing the performance of the algorithms from Section 3, as well as the context $(M, M, \neq)$ with $|M| = 18$, which is special in that every subset of M is closed (and hence there are no valid implications). Both algorithms have been tested in conjunction with each of the three procedures for computing closures (Algorithms 1–3). The results are presented in Table 4 and Fig. 4. It can be seen that Algorithm 6 indeed improves on the performance of Algorithm 5. Among the three algorithms computing the closure, the simpler Algorithm 1 is generally more efficient, even though, in our implementation, we do not perform the initialization

Table 4 Time (in seconds) for building the canonical bases of artificial contexts

Context	# intents	# pseudo-	Algorithm					
		intents	$5+1$	$5+2$	$5+3$	$6+1$	$6+2$	$6+3$
$100 \times 30, 4$	307	557	0.0088	0.0145	0.0119	0.0044	0.0065	0.0059
$10 \times 100, 25$	129	380	0.0330	0.0365	0.0431	0.0073	0.0150	0.0169
$100 \times 50, 4$	251	1,115	0.0442	0.0549	0.0617	0.0138	0.0152	0.0176
$10 \times 100, 50$	559	546	0.0542	0.1312	0.1506	0.0382	0.0932	0.0954
$20 \times 100, 25$	716	2,269	0.3814	0.3920	0.7380	0.1219	0.1312	0.2504
$50 \times 100, 10$	420	3,893	1.1354	0.7291	1.6456	0.1640	0.1003	0.2299
$900 \times 100, 4$	2,472	7,994	4.6313	2.7893	6.3140	1.5594	0.8980	2.0503
$20 \times 100, 50$	12,394	8,136	7.3097	8.1432	14.955	5.1091	6.0182	10.867
$(M, M, \neq)$	262,144	0	0.1578	0.3698	0.1936	0.1333	0.2717	0.1656

A context (G, M, I) is referred to as "$m \times n, k$" if $|G| = m$, $|M| = n$, and $\{g\}' = k$ for each $g \in G$

Fig. 4 Time (in seconds) for building the canonical bases of contexts from Table 3



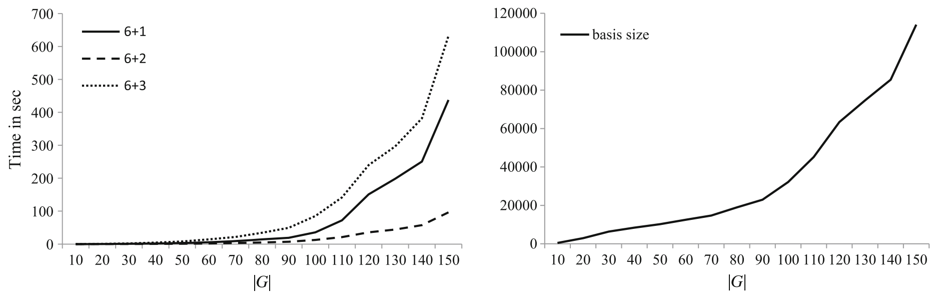


Fig. 5 Time (in seconds) taken by the optimized algorithm for building the canonical bases of contexts with $|G| = 10..150$, $|M| = 150$, and density 0.15 (*left*); the number of implications in the canonical bases of these contexts (*right*)

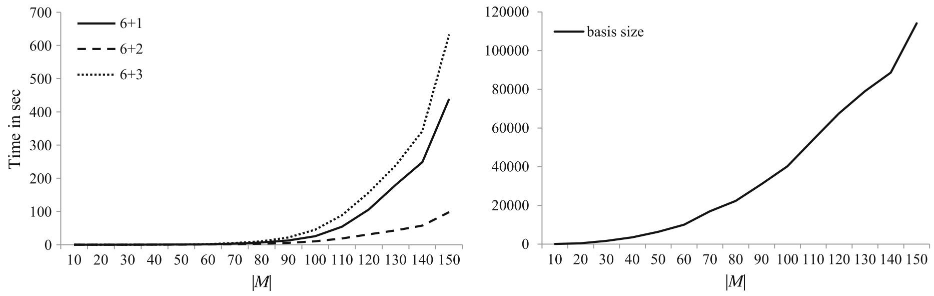


Fig. 6 Time (in seconds) taken by the optimized algorithm for building the canonical bases of contexts with $|G| = 150$, $|M| = 10..150$, and density 0.15 (*left*); the number of implications in the canonical bases of these contexts (*right*)

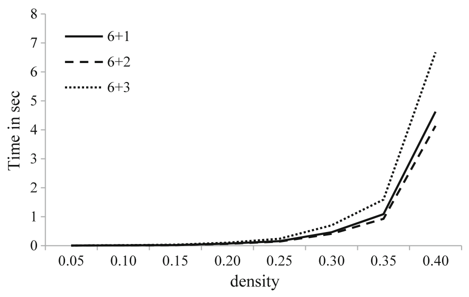
step of Algorithms 2 and 3 from scratch each time we need to compute a closure of a new set; instead, we reuse the previously constructed counters and implication lists and update them incrementally with the addition of each new implication.

However, results presented in Fig. 5 suggest that the asymptotic complexity of LinClosure gives it an advantage over the other algorithms on larger contexts or, rather, on contexts with larger implication bases. On the left-hand side of Fig. 5, we show the time taken by the optimized algorithm (Algorithm 6) in combination with Algorithms 1–3 to build the canonical basis of randomly generated contexts with 150 attributes and the number of objects varying from 10 to 150. In these contexts, the probability of gIm is 0.15 for each $g \in G$ and $m \in M$; thus, the *density* of the contexts is 0.15.[3] On the right-hand side, we show the sizes of the canonical bases of these contexts. One can see that, in this experiment, the size of the basis grows roughly quadratically with the number of objects, and this has a crucial effect on the relative performance of the algorithms, with LinClosure being the most efficient. The situation is similar when we fix $|G|$ and the density and allow $|M|$ to grow (see Fig. 6) and when we fix $|G|$ and $|M|$ and vary the density (see Fig. 7).

5 Other approaches to computing the basis

In our future work, we plan to extend the comparison to algorithms generating the Duquenne–Guigues basis in a different (non-lectic) order, in particular, to incremental [24] and divide-and-conquer [30] approaches. In addition, we are going to consider algorithms that generate other implication covers [8, 23, 29, 31]. Of particular interest is the cover based on a subset of proper premises [15] introduced in [28]. Such covers can be used as an intermediate step in the computation of the Duquenne–Guigues basis. If the number of intents is much larger than the number of pseudo-intents, this two-step approach may be more efficient than direct generation

[3]We do not show the time for Algorithm 5: it is just too slow here. For example, on a context with $|G| = 150$, $|M| = 130$ and density of 0.15, Algorithm 5 in combination with Closure needs 680 seconds, whereas Algorithm 6 (also in combination with Closure) terminates in only 180 s (see Fig. 6).



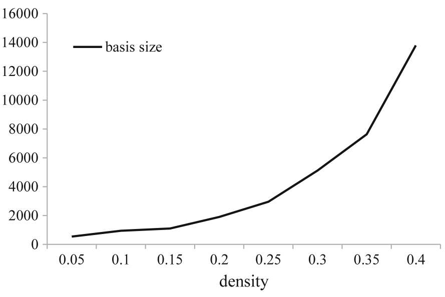


Fig. 7 Time (in seconds) taken by the optimized algorithm for building the canonical bases of contexts with $|G| = |M| = 50$, and density 0.05..0.4 (*left*); the number of implications in the canonical bases of these contexts (*right*)

of the Duquenne–Guigues basis with Algorithms 5 or 6, which produce all intents as a side effect.

Any such cover $\mathcal{L}$ can be transformed into the basis of minimum cardinality in time quadratic in the size of $\mathcal{L}$ [10, 22] using Algorithm 7. This may take long if the size of the cover is significantly bigger than that of the basis. An alternative approach emerges from the work carried out within a well-studied machine learning paradigm known as "learning with queries" [1]. Rather than working with a fixed dataset, the learning model assumes one or several oracles that are capable of answering certain types of questions. In [2], an algorithm Horn1 for learning a Horn formula—that is, essentially, a set of implications $\mathcal{L}_*$—is proposed that uses two types of oracles. In terms of formal concept analysis, these oracles can be described as follows. The *membership* oracle checks if a given attribute set A is closed under the implications in $\mathcal{L}_*$, i.e., if $\mathcal{L}_*(A) = A$. The *equivalence* oracle receives another set $\mathcal{L}$ of implications and replies whether this set is equivalent to $\mathcal{L}_*$; if not, it produces a counterexample, i.e., an attribute set X such that $X = \mathcal{L}_*(X) \neq \mathcal{L}(X)$ (a *positive counterexample*) or $X = \mathcal{L}(X) \neq \mathcal{L}_*(X)$ (a *negative counterexample*).

Algorithm 7 Minimal Cover($\mathcal{L}$)

Input: A set $\mathcal{L}$ of implications over a set M.
Output: Transforms $\mathcal{L}$ into its Duquenne–Guigues basis.

```
for all $A \to B \in \mathcal{L}$ do
    $\mathcal{L} := \mathcal{L} \setminus \{A \to B\}$
    $B := \mathcal{L}(A \cup B)$
    $\mathcal{L} := \mathcal{L} \cup \{A \to B\}$
for all $A \to B \in \mathcal{L}$ do
    $\mathcal{L} := \mathcal{L} \setminus \{A \to B\}$
    $A := \mathcal{L}(A)$
    if $A \neq B$ then
        $\mathcal{L} := \mathcal{L} \cup \{A \to B\}$
```

The algorithm runs in time polynomial in the size of $\mathcal{L}_*$ (assuming that every equivalence or membership query gets answered in constant time); i.e., it is *output-polynomial*. The algorithm starts with the empty set $\mathcal{L}$ of implications and proceeds until a positive answer is obtained from an equivalence query. If a negative example X is received instead, the algorithm uses membership queries to find an implication $A \to B$ in the current set $\mathcal{L}$ such that $A \cap X \neq A$ is not a model of the target set $\mathcal{L}_*$ of implications. If such an implication is found, the implication $A \to B$ is replaced by $A \cap X \to B$; otherwise, a new implication $X \to M$ is added to $\mathcal{L}$. This ensures that X is no longer a model of $\mathcal{L}$. When a positive counterexample X is obtained from an equivalence query, every implication $A \to B$ of which X is not a model is replaced by $A \to B \cap X$. We give pseudo-code in Algorithm 8 and refer the reader to [2] for further details.[4]

Algorithm 8 HORN1(M, $equivalent(\cdot)$, $member(\cdot)$)

Input: An equivalence and a membership oracles for a set of implications $\mathcal{L}_*$ over an attribute set M.
Output: The Duquenne–Guigues basis of $\mathcal{L}_*$.

```
$\mathcal{L} := \varnothing$
while $equivalent(\mathcal{L})$ returns a counterexample $X$ do
    if $X = \mathcal{L}(X)$ then                                {negative counterexample}
        $found$ := false
        for all $A \to B \in \mathcal{L}$ do
            $C := A \cap X$
            if $A \neq C$ and not $member(C)$ then
                $\mathcal{L} := \mathcal{L} \setminus \{A \to B\}$
                $\mathcal{L} := \mathcal{L} \cup \{C \to B\}$
                $found$ := true
                exit for
        if not $found$ then
            $\mathcal{L} := \mathcal{L} \cup \{X \to M\}$
    else                                                        {positive counterexample}
        for all $A \to B \in \mathcal{L}$ such that $A \subseteq X$ and $B \not\subseteq X$ do
            $\mathcal{L} := \mathcal{L} \setminus \{A \to B\}$
            $\mathcal{L} := \mathcal{L} \cup \{A \to B \cap X\}$
return $\mathcal{L}$
```

In a recent paper [3], it is shown that this algorithm always produces the Duquenne–Guigues basis of the target set $\mathcal{L}_*$ no matter what examples are received from the equivalence queries. Thus, Algorithm 8 can be used to build the Duquenne–Guigues basis of a formal context $\mathbb{K}$ if we find a way to efficiently simulate the oracles. Simulating the membership oracle is easy: this amounts to checking whether an attribute set is closed in $\mathbb{K}$. Positive counterexamples to equivalence queries are also easy to obtain: if there is such a counterexample, there is one among object intents of $\mathbb{K}$. Generating negative counterexamples, i.e., sets closed under the current implication set $\mathcal{L}$ but not in the context $\mathbb{K}$, seems much harder. It is not clear

[4]Note that our presentation differs slightly from the one in [2], where the discussion is in terms of Horn clauses. This includes clauses without positive literals, which cannot be represented by implications of the sort we consider here.

whether this can be done in such a way that the resulting algorithm remains output-polynomial. Nevertheless, one possible approach is to generate proper premises (or, for that matter, premises of any implication cover of $\mathbb{K}$) one by one and treat each as a potential negative counterexample; obviously, no other negative examples will be needed. It may turn out more time-efficient (and more space-efficient) than first generating the entire cover and then minimizing it with Algorithm 7.

Another approach is to generate negative counterexamples randomly. Such a randomized algorithm based on Algorithm 8 is proposed in [17]. The algorithm has a one-sided error: it produces a set $\mathcal{L}$ of implications such that $A = \mathcal{L}(A)$ whenever $A = A''$, but it may happen that $A = \mathcal{L}(A)$ even when $A \neq A''$. However, this error is controllable: computing a set $\mathcal{L}$ of implications, the algorithm needs time polynomial in $|G|$, $|M|$, $\mathcal{L}_*$, $1/\epsilon$, and $1/\delta$, where $\mathcal{L}_*$ is the Duquenne–Guigues basis of $\mathbb{K}$ and $0 < \epsilon, \delta \leq 1$, to ensure with probability $1 - \delta$ that the fraction of sets A such that $A = \mathcal{L}(A) \neq A''$ among all subsets of M is at most ϵ. It remains to be seen whether this appealing theoretical guarantee leads to good performance in practice. In particular, we plan to investigate into how the quality of approximation on various types of contexts depends on the time the algorithm is allowed to run. Even more interesting is whether a sufficiently good approximation can be achieved in a fraction of time required by the algorithms studied in this paper to compute the basis exactly. The quality of approximation may be evaluated either formally, by comparing the closure operator induced by the generated implications with the one given by the context, or with an application in mind. In the latter case, it may be useful to guide the randomization in such a way that (mainly) implications with, e.g., high support or other "nice" properties are produced in the result similarly to how concept lattices are sometimes pruned to keep only the most "interesting" concepts [19, 27].

6 Conclusion

In this paper, we presented optimizations to Ganter's algorithm for computing the Duquenne–Guigues basis of a formal context. We also compared the performance of three algorithms computing the closure of an attribute set with respect to a set of implications. Each of these algorithms can be used as a (frequently called) subroutine while computing the Duquenne–Guigues basis. We tested them in conjunction with Ganter's algorithm and its optimized version. In all combinations, the optimized version has a significant performance advantage over the original algorithm. Of the three procedures for closure computation, LinClosure, which features the best theoretical complexity, is generally inferior to the naïve algorithm in application to real data. Nevertheless, our experiments with randomly generated contexts show that, when we fix two of the three input-size parameters, $|G|$, $|M|$, and $|I|$, and allow the third one to grow, LinClosure becomes increasingly more efficient than the other two algorithms. Whether this asymptotic behavior can be observed on large real-life datasets or it is only a result of randomness in data is a matter of further research and experiment.

Acknowledgements The second author was partly supported by the Basic Research Program of the National Research University Higher School of Economics, project "Mathematical models, algorithms, and software tools for knowledge discovery in big data given by relational and textual collections", 2013.

References

1. Angluin, D.: Queries and concept learning. Mach. Learn. **2**, 319–342 (1988)
2. Angluin, D., Frazier, M., Pitt, L.: Learning conjunctions of Horn clauses. Mach. Learn. **9**, 147–164 (1992)
3. Arias, M., Balcázar, J.L.: Construction and learnability of canonical Horn formulas. Mach. Learn. **85**(3), 273–297 (2011)
4. Armstrong, W.W.: Dependency structures of data base relationships. In: Proc. IFIP Congress, pp. 580–583 (1974)
5. Baader, F., Ganter, B., Sertkaya, B., Sattler, U.: Description logic knowledge bases using formal concept analysis. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), pp. 230–235 (2007)
6. Babin, M.A., Kuznetsov, S.O.: Recognizing pseudo-intents is coNP-complete. In: Kryszkiewicz, M., Obiedkov, S. (eds.) Proceedings of the 7th International Conference on Concept Lattices and Their Applications, pp. 294–301. University of Sevilla, Spain (2010)
7. Beeri, C., Bernstein, P.: Computational problems related to the design of normal form relational schemas. ACM Trans. Database Syst. **4**(1), 30–59 (1979)
8. Bertet, K., Monjardet, B.: The multiple facets of the canonical direct unit implicational basis. Theor. Comput. Sci. **411**(22–24), 2155–2166 (2010)
9. Blake, C., Merz, C.: UCI repository of machine learning databases (1998). http://archive.ics.uci.edu/ml
10. Day, A.: The lattice theory of functional dependencies and normal decompositions. Int. J. Algebra Comput. **2**, 409–431 (1992)
11. Demming, R., Duffy, D.: Introduction to the Boost C++ Libraries. Datasim Education Bv (2010). See http://www.boost.org. Accessed 3 May 2013
12. Distel, F., Sertkaya, B.: On the complexity of enumerating pseudo-intents. Discrete Appl. Math. **159**, 450–466 (2011)
13. Ganter, B.: Two basic algorithms in concept analysis. Preprint 831, Technische Hochschule Darmstadt, Germany (1984)
14. Ganter, B.: Attribute exploration with background knowledge. Theor. Comput. Sci. **217**(2), 215–233 (1999)
15. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin (1999)
16. Guigues, J.L., Duquenne, V.: Familles minimales d'implications informatives resultant d'un tableau de donnees binaires. Math. Sci. Hum. **95**(1), 5–18 (1986)
17. Kautz, H., Kearns, M., Selman, B.: Horn approximations of empirical data. Artif. Intell. **74**(1), 129–145 (1995)
18. Khardon, R.: Translating between Horn representations and their characteristic models. J. Artif. Intell. Res. (JAIR) **3**, 349–372 (1995)
19. Klimushkin, M., Obiedkov, S., Roth, C.: Approaches to the selection of relevant concepts in the case of noisy data. In: Kwuida, L., Sertkaya, B. (eds.) Formal Concept Analysis, Lecture Notes in Computer Science, vol. 5986, pp. 255–266. Springer, Berlin/Heidelberg (2010)
20. Kuznetsov, S., Obiedkov, S.: Comparing performance of algorithms for generating concept lattices. J. Exp. Theor. Artif. Intell. **14**(2/3), 189–216 (2002)
21. Kuznetsov, S.O., Obiedkov, S.: Some decision and counting problems of the Duquenne–Guigues basis of implications. Discrete Appl. Math. **156**(11), 1994–2003 (2008)
22. Maier, D.: The theory of relational databases. Computer software engineering series. Computer Science Press, Rockville (1983)
23. Mannila, H., Räihä, K.J.: The design of relational databases. Addison-Wesley Longman Publishing Co., Inc., Boston, MA (1992)
24. Obiedkov, S., Duquenne, V.: Attribute-incremental construction of the canonical implication basis. Ann. Math. Artif. Intell. **49**(1–4), 77–99 (2007)
25. Obiedkov, S., Kourie, D., Eloff, J.: Building access control models with attribute exploration. Comput. Secur. **28**(1–2), 2–7 (2009)
26. Reeg, S., Wei, W.: Properties of Finite Lattices. Diplomarbeit, TH Darmstadt (1990)
27. Roth, C., Obiedkov, S., Kourie, D.G.: On succinct representation of knowledge community taxonomies with formal concept analysis. Int. J. Found. Comput. Sci. **19**(2), 383–404 (2008)

28. Ryssel, U., Distel, F., Borchmann, D.: Fast computation of proper premises. In: Int. Conf. on Concept Lattices and Their Applications, pp. 101–113. INRIA Nancy–Grand Est and LORIA, France (2011)
29. Taouil, R., Bastide, Y.: Computing proper implications. In: Proc. ICCS-2001 International Workshop on Concept Lattices-Based Theory, Methods and Tools for Knowledge Discovery in Databases, pp. 290–303 (2001)
30. Valtchev, P., Duquenne, V.: On the merge of factor canonical bases. In: Medina, R., Obiedkov, S. (eds.) ICFCA, Lecture Notes in Computer Science, vol. 4933, pp. 182–198. Springer, New York (2008)
31. Wild, M.: Computations with finite closure systems and implications. In: Computing and Combinatorics, pp. 111–120 (1995)
32. Yevtushenko, S.A.: System of data analysis "Concept Explorer" (in Russian). In: Proceedings of the 7th National Conference on Artificial Intelligence KII-2000, pp. 127–134. Russia (2000). http://conexp.sourceforge.net/. Accessed 3 May 2013