LinCbO: fast algorithm for computation of the Duquenne-Guigues basis

Radek Janostik, Jan Konecny*, Petr Krajča
Dept. Computer Science, Palacký University Olomouc
17. listopadu 12, CZ–77146 Olomouc
Czech Republic

Abstract
We propose and evaluate a novel algorithm for computation of the Duquenne-Guigues basis which combines Close-by-One and LinClosure algorithms. This combination enables us to reuse attribute counters used in LinClosure and speed up the computation. Our experimental evaluation shows that it is the most efficient algorithm for computation of the Duquenne-Guigues basis.

Keywords: non-redundancy; attribute implications; minimalization; closures.

1. Introduction

Formal Concept Analysis [13, 11] (FCA) has two main outputs: (i) hierarchy of formal concepts, called a concept lattice, in the input data and (ii) a non-redundant system of attribute implications, called a basis, describing the input data. For both of these outputs, closure systems are the fundamental structures behind the related theory and algorithms.

Many algorithms for computing closure systems exist [11, 19]. Among the most efficient algorithms are variants of Kuznetsov’s Close-by-One (CbO) [17], namely Outrata & Vychodil’s FCbO [22] and Andrews’s In-Close family of algorithms [1, 2, 3, 4, 5]. These are commonly used for enumeration of formal concepts, as both their parts, extents and intents, form a closure systems.

When considering systems of attribute implications, pseudo-intents play an important role, since they derive the minimal basis, called the Duquenne-Guigues basis or canonical basis [15]. The pseudo-intents, together with

*Corresponding author

Email addresses: radek.janostik@upol.cz (Radek Janostik),
jan.konecny@upol.cz (Jan Konecny), petr.krajca@upol.cz (Petr Krajča)
intents of formal concepts, form a closure system. Enumerating all pseudo-intents (together with intents) is more challenging as it requires a particular restriction of the order of the computation and the results on complexity are all but promising [18]. There are basically two main approaches for this task: NextClosure by Ganter [14, 13], and the incremental approach by Obiedkov and Duquenne [21].

We present a new approach based on the CbO algorithm and LinClosure [20]. We show that in our approach, LinClosure is able to reuse attribute counters from previous computations. This makes it work very fast, as our experiments show.

The rest of the paper has the following structure: First, we recall basic notions of FCA (Section 2.1), closure operators (Section 2.2), bases of attribute implications (Section 2.3), the algorithm CbO (Section 2.4), and the algorithm LinClosure (Section 2.5). Second, we introduce our approach, which includes CbO with changed sweep order (Section 3.1) and improvements previously introduced into NextClosure in [6] (Section 3.2). Most importantly, we describe a feature which enables LinClosure to reuse the attribute counters (Section 3.3). Then, we experimentally evaluate the resulting algorithm (Section 4) and discuss our observations (Section 4.3). Finally, we summarize our conclusions and present ideas for further research (Section 5).

2. Preliminaries

Here, we recall notions used in the rest of the paper.

2.1. Formal concept analysis

An input to FCA is a triplet \( \langle X, Y, I \rangle \), called a formal context, where \( X, Y \) are non-empty sets of objects and attributes respectively, and \( I \) is a binary relation between \( X \) and \( Y \). The presence of an object-attribute pair \( \langle x, y \rangle \) in the relation \( I \) means that the object \( x \) has the attribute \( y \).

Finite contexts are usually depicted as tables, in which rows represent objects in \( X \), columns represent attributes in \( Y \), ones in its entries mean that the corresponding object-attribute pair is in \( I \).

The formal context \( \langle X, Y, I \rangle \) induces so-called concept-forming operators:

\[ \uparrow : 2^X \to 2^Y \]

assigns to a set \( A \) of objects the set \( A^\uparrow \) of all attributes shared by all the objects in \( A \).

\[ \downarrow : 2^Y \to 2^X \]

assigns to a set \( A \) of attributes the set \( A^\downarrow \) of all objects shared by all the attributes in \( A \).
\[ \dagger : 2^Y \rightarrow 2^X \] assigns to a set \( B \) of attributes the set \( B^\dagger \) of all objects which share all the attributes in \( B \).

Formally, for all \( A \subseteq X, B \subseteq Y \) we have

\[
A^\dagger = \{ y \in Y \mid \forall x \in A : \langle x, y \rangle \in I \}, \\
B^\dagger = \{ x \in X \mid \forall y \in B : \langle x, y \rangle \in I \}.
\]

Fixed points of the concept-forming operators, i.e. pairs \( \langle A, B \rangle \in 2^X \times 2^Y \) satisfying \( A^\dagger = B \) and \( B^\dagger = A \), are called formal concepts. The sets \( A \) and \( B \) in a formal concept \( \langle A, B \rangle \) are called the extent and the intent, respectively.

The set of all intents in \( \langle X, Y, I \rangle \) is denoted by \( \text{Int}(X, Y, I) \).

An attribute implication is an expression of the form \( L \Rightarrow R \) where \( L, R \subseteq Y \) are sets of attributes.

We say that \( L \Rightarrow R \) is valid in a set of attributes \( M \subseteq Y \) if

\[
L \subseteq M \implies R \subseteq M.
\]

The fact that \( L \Rightarrow R \) is valid in \( M \) is written as \( \| L \Rightarrow R \|_M = 1 \).

We say that \( L \Rightarrow R \) is valid in a context \( \langle X, Y, I \rangle \) if it is valid in every object intent \( \{x\}^\dagger \), i.e.

\[
\| L \Rightarrow R \|_{\{x\}^\dagger} = 1 \quad \forall x \in X.
\]

A set of attribute implications is called a theory.

A set of attributes \( M \) is called a model of theory \( T \) if every attribute implication in \( T \) is valid in \( M \). The set of all models of \( T \) is denoted \( \text{Mod}(T) \), i.e.

\[
\text{Mod}(T) = \{ M \mid \forall L \Rightarrow R \in T : \| L \Rightarrow R \|_M = 1 \}.
\]

2.2. Closure systems and closure operators

A closure system in a set \( Y \) is any system \( S \) of subsets of \( Y \) which contains \( Y \) and is closed under arbitrary intersections.

A closure operator on a set \( Y \) is a mapping \( c : 2^Y \rightarrow 2^Y \) satisfying for each \( A, A_1, A_2 \subseteq Y \):

\[
A \subseteq c(A) \quad (1) \\
A_1 \subseteq A_2 \implies c(A_1) \subseteq c(A_2) \quad (2) \\
c(A) = c(c(A)) \quad (3).
\]

The closure systems and closure operators are in one-to-one correspondence. Specifically, for a closure system \( S \) in \( Y \), the mapping \( c_S : 2^Y \rightarrow 2^Y \) defined by

\[
c_S(A) = \bigcup \{ B \in S \mid A \subseteq B \}
\]
is a closure operator. Conversely, for a closure operator $c$ on $Y$, the set
\[ \mathcal{S}_c = \{ A \in 2^Y \mid c(A) = A \} \]
is a closure system. Furthermore, $\mathcal{S}_{c_s} = \mathcal{S}$ and $c_{\mathcal{S}_c} = c$.

For a formal context $\langle X, Y, I \rangle$, the set $\text{Int}(X, Y, I)$ of its intents is a closure system. The corresponding closure operator, $c_{\text{Int}(X,Y,I)}$, is equal to the composition $c_\uparrow$ of concept-forming operators.

For any theory $\mathcal{T}$, the set $\text{Mod}(\mathcal{T})$ of its models is a closure system. The corresponding closure operator, $c_{\text{Mod}(\mathcal{T})}$, is equal to the following operator $c_{\mathcal{T}}$. For $Z \subseteq Y$ and theory $\mathcal{T}$, put
\[
\begin{align*}
1. & \quad Z^\mathcal{T} = Z \cup \bigcup \{ R \mid L \Rightarrow R \in T, L \subseteq Z \}, \\
2. & \quad Z^{T_0} = Z, \\
3. & \quad Z^{T_n} = (Z^{T_{n-1}})^\mathcal{T}.
\end{align*}
\]
Define operator $c_{\mathcal{T}} : 2^Y \to 2^Y$ by
\[ c_{\mathcal{T}}(Z) = \bigcup_{n=0}^{\infty} Z^{T_n}. \]

2.3. Bases, Duquenne-Guigues basis and its computation

A theory $\mathcal{T}$ is called

- complete in $\langle X, Y, I \rangle$ if $\text{Mod}(\mathcal{T}) = \text{Int}(X, Y, I)$;
- a basis of $\langle X, Y, I \rangle$ if no proper subset of $\mathcal{T}$ is complete in $\langle X, Y, I \rangle$.

A set $P \subseteq Y$ of attributes is called a pseudo-intent if it satisfies the following conditions:

(i) it is not an intent, i.e. $P \uparrow \uparrow \neq P$;
(ii) for all smaller pseudo-intents $P_0 \subset P$, we have $P_0 \uparrow \uparrow \subset P$.

**Theorem 1.** Let $\mathcal{P}$ be a set of all pseudo-intents of $\langle X, Y, I \rangle$. The set
\[ \{ P \Rightarrow P \uparrow \mid P \in \mathcal{P} \} \]
is a basis of $\langle X, Y, I \rangle$. Additionally, it is a minimal basis in terms of the number of attribute implications.

The basis from Theorem 1 is called the Duquenne-Guigues basis.

Let $\mathcal{P}$ be a set of all pseudo-intents of $\langle X, Y, I \rangle$. The union $\text{Int}(X, Y, I) \cup \mathcal{P}$ is a closure system on $Y$.

The corresponding closure operator $\tilde{c}_{\mathcal{T}}$ is given as follows. For $Z \subseteq Y$ and theory $\mathcal{T}$, put
1. $Z^T = Z \cup \bigcup \{ R \mid L \Rightarrow R \in T, L \subset Z \}$,
2. $Z^{T_0} = Z$,
3. $Z^{T_n} = (Z^{T_{n-1}})^T$.

Define operator $\tilde{c}_T : 2^Y \rightarrow 2^Y$ by

\[
\tilde{c}_T(Z) = \bigcup_{n=0}^{\infty} Z^{T_n}.
\] (4)

Note that the definition of $\tilde{c}_T$ differs from the definition of $c_T$ in Section 2.2 only in the subsethood in item 1 – the operator $c_T$ allows equality in this item while $\tilde{c}_T$ does not. In what follows, we use the shortcut $Z^*$ for $\tilde{c}_T(Z)$.

The algorithm which follows the above definition is called the naïve algorithm. There are more sophisticated ways to compute closures, like LinClosure [20] and Wild’s closure [24].

To compute the closure system $\text{Int}(X, Y, I) \cup \mathcal{P}$ using the above closure operator, the intents and pseudo-intents must be enumerated in an order which extends the subsethood; i.e.

\[
C_1 \subseteq C_2 \text{ implies } C_1 \leq C_2 \text{ for all } C_1, C_2 \in \text{Int}(X, Y, I) \cup \mathcal{P}.
\] (5)

The lexic order satisfies this condition; that is why NextClosure [13] is most frequently used for this task.

2.4. Close-by-One

We assume a closure operator $c$ on set $Y = \{1, 2, \ldots, n\}$. Whenever we write about lower attributes or higher attributes, we refer to the natural ordering of the numbers in $Y$.

We start the description of CbO with a basic algorithm for generating all closed sets. The basic algorithm traverses the space of all subsets of $Y$, each subset is checked for closedness and is outputted. This approach is quite inefficient as the number of closed subsets is typically significantly smaller than the number of all subsets.

The algorithm is given by a recursive procedure $\text{GenerateFrom}$, which accepts two arguments:

- $B$ – the set of attributes, from which new sets will be generated.
- $y$ – the auxiliary argument to remember the highest attribute in $B$.

The procedure first checks the input set $B$ for closedness and prints it if it is closed (lines 1,2). Then, for each attribute $i$ higher than $y$:

- a new set is generated by adding the attribute $i$ into the set $B$ (line 4);
Figure 1: Tree of all subsets of $\{1, 2, 3, 4\}$. Each node represents a unique set containing all elements in the path from the node to the root. The dotted arrows and small numbers represent the sweep performed by the CbO algorithm.

- the procedure recursively calls itself to process the new set (line 5).

The procedure is initially called with an empty set and zero as its arguments.

The basic algorithm represents a depth-first sweep through the tree of all subsets of $Y$ (see Fig. 1) and printing the closed ones.

In the tree of all subsets (Fig. 1), each node is a superset of its predecessors. We can use the closure operator $^\uparrow$ to skip non-closed sets. In other words, to make jumps in the tree to closed sets only. Instead of simply adding an element to generate a new subset

$$D \leftarrow B \cup \{i\},$$
CbO adds the element and then closes the set
\[ D \leftarrow c(B \cup \{i\}). \] (6)

We need to distinguish the two outcomes of the closure (6). Either

- the closure contains some attributes lower than \(i\) which are not included in \(B\), i.e.
  \[ D_i \neq B_i \]
  where \( D_i = D \cap \{1, \ldots, i-1\}, B_i = B \cap \{1, \ldots, i-1\} \);
- or it does not, and we have
  \[ D_i = B_i. \]

The jumps with \( D_i \neq B_i \) are not desirable because they land on a closed set which was already processed or will be processed later (depending on the direction of the sweep). CbO does not perform such jumps. The check of the condition \( D_i = B_i \) is called a canonicity test.

**Algorithm 2: Close-by-One**

```python
def CbOStep(B, y):
    input : B - closed set
            y - last added attribute
    print(B)
    for i in \{y + 1, \ldots, n\} \ B do
        D \leftarrow c(B \cup \{i\})
        if D_y = B_y then
            CbOStep(D, i)
CbOStep(\emptyset, 0)
```

One can see the pseudocode of CbO in Algorithm 2.

We describe the differences from the basic algorithm:

- The argument \(B\) is a closed set, therefore, the procedure `GenerateFrom` can print it directly without testing (line 1).
- In the loop, we skip elements already present in \(B\) (line 2).
- The recursive invocation is made only if the new closed set \(D\) passes the canonicity test (lines 3,4).
2.5. LinClosure

LinClosure (Algorithm 3) \cite{7,20} accepts a set \( B \) of attributes for which it computes the \( \mathcal{T} \)-closure \( \tilde{c}_\mathcal{T}(B) \). The theory \( \mathcal{T} \) is considered to be a global variable. It starts with a set \( D \) containing all elements of \( B \) (line 1). If there is an attribute implication in \( \mathcal{T} \) with empty left side, the \( D \) is unified with its right side (lines 2,3). LinClosure associates a counter \( \text{count}[L \Rightarrow R] \) with each \( L \Rightarrow R \in \mathcal{T} \) initializing it with the size \( |L| \) of its left side (lines 4,5). Also, each attribute \( y \in Y \) is linked to a list of the attribute implications that have \( y \) in their left sides (lines 6,7). Then, the set \( Z \) of attributes to be processed is initialized as a copy of the set \( D \) (line 8). While there are attributes in \( Z \), the algorithm chooses one of them (\( \text{min} \) in the pseudocode, line 10), removes it from \( Z \) (line 11) and decrements counters of all attribute implication linked to it (lines 12,13). If the counter of any attribute implication \( L \Rightarrow R \) is decreased to 0, new attributes from \( R \) are added to \( D \) and to \( Z \).

We are going to use the algorithm LinClosure in CbO. CbO drops the resulting closed set if it fails the canonicity test (Algorithm 2, lines 4,5). Therefore, we can introduce a feature which stops the computation whenever an attribute which would cause the fail is added into the set. To do that, we add a new input argument, \( y \), having the same role as in CbO; i.e. the last attribute added into the set (Algorithm 4). Then, whenever new attributes are added to the set, we check whether any of them is lower than \( y \). If so, we stop the procedure and return information that the canonicity test would fail (lines 16–18).

3. LinCbO: CbO-based algorithm for computation of the Duquenne-Guigues basis

In this section, we describe the algorithm LinCbO. Its foundation is CbO (Algorithm 2) with LinClosure (Algorithm 3). We explain changes in the CbO algorithm: a change of sweep order makes the algorithms work, and the rest of the changes improve efficiency of the algorithms.

3.1. Sweep order

In the previous section, we presented CbO as the left first sweep through the tree of all subsets. This is how it is usually described. In ordinary settings, there is no need to follow a particular order of sweep. However, our purpose is to compute intents and pseudo-intents using the closure operator

\[ \text{This needs to be done just once and it is usually done outside the LinClosure procedure.} \]

\[ \text{This feature is also utilized in \cite{6}.} \]
Algorithm 3: LinClosure

```python
def LinClosure(B):
    input : B – set of attributes
    1. D ← B
    2. if ∃∅ ⇒ R ∈ T for some R then
       3. D ← D ∪ R
    4. for all L ⇒ R ∈ T do
       5. count[L ⇒ R] ← |L|
       6. for all a ∈ L do
          7. add L ⇒ R to list[a]
    8. Z ← D
    9. while Z ≠ ∅ do
       10. m ← min(Z)
       11. Z ← Z\{m}
       12. for all L ⇒ R ∈ list[m] do
          13. count[L ⇒ R] ← count[L ⇒ R] − 1
          14. if count[L ⇒ R] = 0 then
             15. add ← R\D
             16. D ← D ∪ add
             17. Z ← Z ∪ add
       18. return D
```

Algorithm 4: LinClosure with an early stop

```
def LinClosureES(B, y):
    input : B – set of attributes
            y – last attribute added to B

    D ← B
    if ∃∅ ⇒ R ∈ T for some R then
        D ← D ∪ R
    for all L ⇒ R ∈ T do
        count[L ⇒ R] ← |L|
        for all a ∈ L do
            add L ⇒ R to list[a]
    Z ← D

    while Z ≠ ∅ do
        m ← min(Z)
        Z ← Z\{m}
        for all L ⇒ R ∈ list[m] do
            count[L ⇒ R] ← count[L ⇒ R] − 1
            if count[L ⇒ R] = 0 then
                add ← R\D
                if min(add) < y then
                    return fail
                else
                    D ← D ∪ add
            Z ← Z ∪ add

    return D
```
Figure 2: Tree of all subsets of \{1, 2, 3, 4\}. Each node represents a unique set containing all elements in the path from the node to the root. The dotted arrows and small numbers represent the sweep performed by the CbO algorithm with right depth-first sweep.

\( \tilde{c}_T \) \( a \). For this, we need to utilize an order which extends the subsethood, i.e. \( a \). The right depth-first sweep through the tree of all subsets satisfies this condition (see Fig. 2). Observe that with the right depth-first sweep, we obtain exactly the lexicographic order, i.e. the same order in which NextClosure explores the search space.

3.2. NextClosure’s improvements

The following improvements were introduced to NextClosure \[6\] and the incremental approach \[21\] for computation of pseudo-intents. We incorporated them to the CbO algorithm.

After the algorithm computes \( B' \), the implication \( B'^* \rightarrow B^\uparrow \) is added to \( T \), provided \( B'^* \) is a pseudo-intent, i.e. \( B'^* \neq B^\uparrow \).

Note that there exists the smallest \( \tilde{c}_T \)-closed set larger than \( B' \) and it is the intent \( B^* \) (\( = B^\uparrow \)). Consider the following two cases:

\( o_1 \) This intent satisfies the canonicity test, i.e. \( B'^{\uparrow y} = B^*_y \), where \( y \) is the last added attribute to \( A \). Then we can jump to this intent.

\( o_2 \) This intent does not satisfy the canonicity test. Thus, we can leave the present subtree.

Now, let us describe the first version of LinCbO (Algorithm 5), which includes the above discussed improvements.

The procedure LinCbO1Step works with the following global variables: an initially empty theory \( T \), and an initially empty list of attribute implication for each attribute. LinCbO1Step accepts two arguments: a set \( B \) of attributes
and the last attribute \( y \) added to \( B \). The set \( B \) is not generally closed (which was the case in Algorithm 2).

The procedure first applies LinClosure with an early stop (Algorithm 4) to compute \( B^* \) (line 1). If \( B^* \) fails the canonicity test (recall that the canonicity test is incorporated in LinClosure with an early stop), the procedure stops (lines 2,3). Then, the procedure computes \( B^* \uparrow \) to check whether \( B^* \) is an intent or pseudo-intent (line 4). If it is a pseudo-intent, a new attribute implication \( B^* \Rightarrow B^* \uparrow \) is added to the initially empty theory \( T \) (line 5). For each attribute in \( B^* \), we update its list by adding the new attribute implication (lines 6 and 7).

Now, as we computed the intent \( B^* \uparrow \), we can apply (o1) or (o2) based on the result of the canonicity test \( B^* \uparrow = B^* \) (line 8) – either we call LinCbO1Step for \( B^* \uparrow \) (line 9) or end the procedure. If \( B^* \) is an intent, we recursively call LinCbO1Step for all sets \( B^* \cup \{i\} \) where \( i \) is higher than the last added attribute \( y \) and is not already present in \( B^* \). To have lexic order, we make the recursive calls in the descending order of \( i \).

The procedure LinCbO1Step is initially called with empty set of attributes and zero representing an invalid last added attribute.

### 3.3. LinClosure with reused counters

Consider theory \( T' \) and theory \( T \) which emerges by adding new attribute implications to \( T' \), i.e. \( T' \subseteq T \). When we compute \( T' \)-closure \( B' \), we can store values of the attribute counters at the end of the LinClosure procedure. Later, when we compute \( T \)-closure of a superset \( B \) of \( B' \), we can initialize the attribute counters of implications from \( T' \) to the stored values instead of the antecedent sizes. Attribute counters for new implications, i.e. those in \( T' \setminus T \), are initialized the usual way. Then, we handle only the new attributes, that is those in \( B \setminus B' \).

We can improve the LinClosure accordingly (Algorithm 6). We describe only the differences from LinClosure with an early stop (Algorithm 4). It accepts two additional arguments: \( Z' \) – the set of new attributes, i.e. those which were not in the \( T \)-closed subset from which we reuse the counters; and prevCount – the previous counters to be reused. We copy the previous counters (lines 4,5) and new attributes \( Z' \) to local variables. Furthermore, we add new attribute implications (lines 6,7).

Note, that in CbO we always make the recursive invocations for supersets of the current set (see Algorithm 5 lines 9 and 12). Therefore, we can easily utilize the LinClosure with reused counters in LinCbO (Algorithm 7). The only difference from the first version (Algorithm 5) is that the procedure LinCbO1Step accepts two additional arguments, which are passed to procedure LinClosureRC (line 1). The two arguments are: the set of new attributes
Algorithm 5: LinCbO1 (CbO for the Duquenne-Guiguès basis, first version)

\[ T \leftarrow \emptyset \]
\[ \text{list}[i] \leftarrow \emptyset \text{ for each } i \in Y \]

\begin{algorithm}
def LinCbO1Step(B, y):
    \textbf{input}: B – set of attributes
    \hspace{1cm} y – last attribute added to B
    \hspace{1cm} \begin{align*}
        & B^* \leftarrow \text{LinClosureES}(B, y) \\
        & \text{if } B^* \text{ is fail then}
        & \hspace{1cm} \text{return} \\
        & \text{if } B^* \neq B^{\uparrow \downarrow} \text{ then}
        & \hspace{1cm} T \leftarrow T \cup \{B^* \Rightarrow B^{\uparrow \downarrow}\}
        & \hspace{1cm} \text{for } i \in B^* \text{ do}
        & \hspace{1.5cm} \text{list}[i] \leftarrow \text{list}[i] \cup \{B^* \Rightarrow B^{\uparrow \downarrow}\}
        & \hspace{1cm} \text{if } B^{\downarrow \uparrow}_y = B^*_y \text{ then}
        & \hspace{1.5cm} \text{LinCbO1Step}(B^{\downarrow \uparrow}_y, y)
        & \text{else}
        & \hspace{1cm} \text{for } i \text{ from } n \text{ down to } y + 1, i \notin B^* \text{ do}
        & \hspace{1.5cm} \text{LinCbO1Step}(B^* \cup \{i\}, i)
    \end{align*}
\end{algorithm}

LinCbO1Step(\emptyset, 0)
Algorithm 6: LinClosure with reused counters

```python
def LinClosureRC(B, y, Z, prevCount):
    input:
        $B$ – set of attributes to be closed
        $y$ – last attribute added to $B$
        $Z'$ – set of new attributes
        $prevCount$ – previous attribute counters from computation $B\setminus Z$
    $D \leftarrow B$
    if $\exists \emptyset \Rightarrow R \in T$ then
        $D \leftarrow D \cup R$
    $count \leftarrow$ copy of $prevCount$
    $Z \leftarrow Z'$
    for $L \Rightarrow R \in T$ not counted in $prevCount$ do
        $count[L \Rightarrow R] \leftarrow |L\setminus B|$
    while $Z \neq \emptyset$ do
        $m \leftarrow \min(Z)$
        $Z \leftarrow Z\setminus\{m\}$
        for $L \Rightarrow R \in list[m]$ do
            $count[L \Rightarrow R] \leftarrow count[L \Rightarrow R] - 1$
            if $count[L \Rightarrow R] = 0$ then
                $add \leftarrow R\setminus D$
                if $\min(add) < y$ then
                    return fail
                    $D \leftarrow D \cup add$
                    $Z \leftarrow Z \cup add$
            return $\langle D, count \rangle$
```

and the previous attribute counters (both initially empty). Recall that the attribute counters are modified by LinClosure. The corresponding arguments are also passed to the recursive invocations of LinCbOStep (lines 9 and 12).

Algorithm 7: LinCbO (CbO for the Duquenne-Guigues basis, final version)

\[
\begin{align*}
\mathcal{T} &\leftarrow \emptyset \\
\text{list}[i] &\leftarrow 0 \quad \text{for each} \quad y \in Y \\
\text{def} \quad \text{LinCbOStep}(B, y, Z, \text{prevCount}): \\
\quad \text{input} : \quad &B - \text{set of attributes} \\
\quad &y - \text{last attribute added to } B \\
\quad &Z - \text{set of new attributes} \\
\quad &\text{prevCount} - \text{attribute counters} \\
\quad &\langle B^*, \text{count} \rangle \leftarrow \text{LinClosureRC}(B, y, Z, \text{prevCount}) \\
\quad &\text{if } B^* \text{ is fail then} \\
\quad &\quad \text{return} \\
\quad &\text{if } B^* \neq B^* \uparrow \text{ then} \\
\quad &\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{B^* \Rightarrow B^* \uparrow\} \\
\quad &\quad \text{for } i \in B^* \text{ do} \\
\quad &\quad \quad \text{list}[i] \leftarrow \text{list}[i] \cup \{B^* \Rightarrow B^* \uparrow\} \\
\quad &\quad \text{if } B^* \uparrow = B_y^* \text{ then} \\
\quad &\quad \quad \text{LinCbOStep}(B^* \uparrow, y, B^* \uparrow \setminus B^*, \text{count}) \\
\quad &\quad \text{else} \\
\quad &\quad \quad \text{for } i \text{ from } n \text{ down to } y + 1, i \notin B^* \text{ do} \\
\quad &\quad \quad \quad \text{LinCbOStep}(B^* \cup \{i\}, \{i\}, \text{count}) \\
\quad &\text{LinCbOStep}(\emptyset, 0, \emptyset, \emptyset)
\end{align*}
\]

4. Experimental Comparison

We compare LinCbO with other algorithms, namely:

- NextClosure with naïve closure (NC1), LinClosure (NC2), and Wild’s closure (NC3).
- NextClosure+, which is NextClosure with the improvements described
in Section 3.2, with the same closures (NC+1, NC+2, NC+3).

• attribute incremental approach [21].

To achieve maximal fairness, we implemented LinCbO into the framework made by Bazhanov & Obiedkov [6]. It contains implementations of all the listed algorithms. In Section 4.1, we also use the same datasets as used by Bazhanov and Obiedkov [6].

All experiments have been performed on a computer with 64 GB RAM, two Intel Xeon CPU E5-2680 v2 (at 2.80 GHz), Debian Linux 10, and GNU GCC 8.3.0. All measurements have been taken ten times and the mean value is presented.

4.1. Batch 1: datasets used in [6]

Bazhanov and Obiedkov [6] use artificial datasets and datasets from UC Irvine Machine Learning Repository [12].

The artificial datasets are named as |X||x||Y|−d, where d is the number of attributes of each object; i.e. |{x}1| = d for each x ∈ X. The attributes are assigned to objects randomly, with exception 18x18-17, where each object misses a different attribute (more exactly, the incidence relation is the inequality).

The datasets from UC Irvine Machine Learning Repository are: Breast-cancer, Breast-w, dbdata0, flare, Post-operative, spect, vote, and zoo. See Table 1 for properties of all the datasets.

In batch 1, LinCbO computes the basis faster than the rest of algorithms; however in most cases the runtimes are very small and differences between them are negligible (see Table 2).

4.2. Batch 2: our collection of datasets

As the runtimes in batch 1 often differ only in a few milliseconds, we tested the algorithm on larger datasets. We used the following datasets from UC Irvine Machine Learning Repository [12]:

• crx – Credit Approval (37 rows containing a missing value were removed),

• shuttle – Shuttle Landing Control,

• magic – MAGIC Gamma Telescope,

4NextClosure and NextClosure+ are called Ganter and Ganter+ in [6].
5Available at https://github.com/yazevnull/fcai
### Table 1: Properties of the datasets in batch 1

| dataset | |X| |Y| |I| |# intents| |# ps.intents|
|---|---|---|---|---|---|---|---|---|
| 100x30-4 | 100 | 30 | 400 | 307 | 557 |
| 100x50-4 | 100 | 50 | 400 | 251 | 1115 |
| 10x100-25 | 10 | 100 | 250 | 129 | 380 |
| 10x100-50 | 10 | 100 | 500 | 559 | 546 |
| 18x18-17 | 18 | 18 | 306 | 262,144 | 0 |
| 20x100-25 | 20 | 100 | 500 | 716 | 2269 |
| 20x100-50 | 20 | 100 | 1000 | 12,394 | 8136 |
| 50x100-10 | 50 | 100 | 500 | 420 | 3893 |
| 900x100-4 | 900 | 100 | 3600 | 2472 | 7994 |
| Breast-cancer | 286 | 43 | 2851 | 9918 | 3354 |
| Breast-w | 699 | 91 | 6974 | 9824 | 10,666 |
| dbdata0 | 298 | 88 | 1833 | 2692 | 1920 |
| flare | 1389 | 49 | 18,062 | 28,742 | 3382 |
| Post-operative | 90 | 26 | 807 | 2378 | 619 |
| spect | 267 | 23 | 2042 | 21,550 | 2169 |
| vote | 435 | 18 | 3856 | 10,644 | 849 |
| zoo | 101 | 28 | 862 | 379 | 141 |

- **bikesharing_-(day|hour)** – Bike Sharing Dataset,
- **kegg** – KEGG Metabolic Reaction Network – Undirected.

We binarized the datasets using nominal (nom), ordinal (ord), and interordinal (inter) scaling, where each numerical feature was scaled to \( k \) attributes with \( k - 1 \) equidistant cutpoints. Categorical features were scaled nominally to a number of attributes corresponding to the number of categories. After the binarization, we removed full columns. Properties of the resulting datasets are shown in Table 3. The naming convention used in Table 3 (and Table 4) is the following: (scaling)\( k \)(dataset). For example, inter10shuttle is the dataset ‘Shuttle Landing Control’ interordinally scaled to 10, using 9 equidistant cutpoints.

For this batch, we included LinCbO1 (Algorithm 5) to show how the reuse of attribute counters influences the performance.

For most datasets, LinCbO works faster than the other algorithms. For the remaining datasets, LinCbO is the second best after the attribute incremental approach (see Table 4). However, we encountered limits of the attribute incremental approach as it runs out of available memory in four cases (denoted by the symbol * in Table 4).
## Table 2: Runtimes in seconds of algorithms generating Duquenne-Guigues basis in batch 1.

| Dataset        | AttnInc | NC1  | NC2  | NC3  | NC+1 | NC+2 | NC+3 | LinCbO |
|----------------|---------|------|------|------|------|------|------|--------|
| 100x30-4       | 0.008   | 0.007| 0.007| 0.01 | 0.004| 0.003| 0.005| 0.002  |
| 100x50-4       | 0.028   | 0.037| 0.024| 0.05 | 0.013| 0.008| 0.016| 0.005  |
| 10x100-25      | 0.015   | 0.015| 0.023| 0.033| 0.007| 0.01  | 0.014| 0.004  |
| 10x100-50      | 0.037   | 0.052| 0.087| 0.112| 0.038| 0.063| 0.081| 0.015  |
| 18x18-17       | 0.337   | 0.096| 0.143| 0.134| 0.111| 0.157| 0.151| 0.148  |
| 20x100-25      | 0.099   | 0.281| 0.165| 0.484| 0.094| 0.061| 0.172| 0.026  |
| 20x100-50      | 0.94    | 5.457| 3.047| 8.898| 3.809| 2.31 | 6.481| 0.675  |
| 50x100-5       | 0.454   | 0.778| 0.253| 1.064| 0.126| 0.047| 0.164| 0.029  |
| 900x100-4      | 2.061   | 3.315| 0.91 | 3.936| 1.15 | 0.317| 1.333| 0.172  |
| Breast-cancer  | 0.121   | 0.295| 0.236| 0.325| 0.231| 0.184| 0.251| 0.055  |
| Breast-w       | 2.856   | 4.674| 3.128| 9.61 | 2.526| 1.67 | 5.155| 0.516  |
| dbdata0        | 0.109   | 0.254| 0.312| 0.43 | 0.158| 0.208| 0.263| 0.049  |
| flare          | 0.622   | 1.006| 1.865| 1.813| 0.92 | 1.661| 1.624| 0.265  |
| Post-operative | 0.014   | 0.015| 0.023| 0.021| 0.013| 0.018| 0.018| 0.009  |
| spect          | 0.142   | 0.407| 0.584| 0.397| 0.388| 0.556| 0.377| 0.097  |
| vote           | 0.054   | 0.062| 0.078| 0.068| 0.059| 0.075| 0.064| 0.024  |
| zoo            | 0.004   | 0.003| 0.005| 0.005| 0.002| 0.004| 0.004| 0.002  |
| dataset          | |X| |Y| |J| # intents | # ps.intents |
|------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| inter10crx       | 653             | 139             | 40,170          | 10,199,818      | 20,108          |
| inter10shuttle   | 43,500          | 178             | 3,567,907       | 38,199,148      | 936             |
| inter3magic      | 19,020          | 52              | 399,432         | 1,006,553       | 4181            |
| inter4magic      | 19,020          | 72              | 589,638         | 24,826,749      | 21,058          |
| inter5bike_day   | 731             | 93              | 24,650          | 3,023,326       | 20,425          |
| inter5crx        | 653             | 79              | 20,543          | 348,428         | 3427            |
| inter5shuttle    | 43,500          | 88              | 1,609,510       | 333,783         | 346             |
| inter6shuttle    | 43,500          | 106             | 2,002,790       | 381,636         | 566             |
| nom10bike_day    | 731             | 100             | 9293            | 52,697          | 29,773          |
| nom10crx         | 653             | 85              | 8774            | 51,078          | 6240            |
| nom10magic       | 19,020          | 102             | 209,220         | 583,386         | 154,090         |
| nom10shuttle     | 43,500          | 97              | 435,000         | 2931            | 810             |
| nom15magic       | 19,020          | 152             | 209,220         | 1,149,717       | 397,224         |
| nom20magic       | 19,020          | 202             | 209,220         | 1,376,212       | 654,028         |
| nom5bike_day     | 731             | 65              | 9293            | 61,853          | 16,296          |
| nom5bike_hour    | 17,379          | 90              | 238,292         | 1,868,205       | 320,679         |
| nom5crx          | 653             | 55              | 8774            | 29,697          | 2162            |
| nom5keg          | 65,554          | 144             | 1,834,566       | 13,262,627      | 42,992          |
| nom5shuttle      | 43,500          | 52              | 435,000         | 1461            | 319             |
| ord10bike_day    | 731             | 93              | 28,333          | 664,713         | 11,795          |
| ord10crx         | 653             | 79              | 37,005          | 1,547,971       | 2906            |
| ord10shuttle     | 43,500          | 88              | 1,849,216       | 97,357          | 279             |
| ord5bike_day     | 731             | 58              | 14,929          | 81,277          | 5202            |
| ord5bike_hour    | 17,379          | 83              | 457,578         | 2,174,964       | 99,691          |
| ord5crx          | 653             | 49              | 19,440          | 139,752         | 973             |
| ord5magic        | 19,020          | 42              | 535,090         | 821,796         | 1267            |
| ord5shuttle      | 43,500          | 43              | 868,894         | 4068            | 119             |
| ord6magic        | 19,020          | 52              | 662,177         | 2,745,877       | 2735            |
Table 4: Runtimes in seconds of algorithms generating Duquenne-Guigues basis in batch 2. The symbol * means that the run could not be completed due to insufficient memory.

| Dataset     | AttInc | NC1    | NC2    | NC3    | NC+1   | NC+2   | NC+3   | LinCbO  | LinCbO1 |
|-------------|--------|--------|--------|--------|--------|--------|--------|---------|---------|
| inter10crx  | 400.292| 2084.12| 17.050.5| 4256.41| 2097.54| 16.817.5| 4193.46| 508.551 | 23.842  |
| inter10shuttle| *     | 18.038.1| 21.268.1| 20.211.9| 17.664.5| 21.035.4| 20.171.9| 15.852.9| 28.373.5|
| inter3magic | 109.178| 106.341| 136.738| 109.133| 107.357| 136.842| 109.428| 26.156  | 74.98   |
| inter4magic | *     | 4029.95 | 9998.74 | 4241.51| 4027.48| 10.023 | 4239.26| 965.353 | 9258.53 |
| inter5bike_day | 72.952| 389.073| 1409.69 | 680.789| 383.537| 1378.89| 670.109| 85.591  | 1589.58 |
| inter5crx   | 5.863  | 16.357 | 56.977 | 25.08  | 16.257 | 56.669 | 24.995 | 3.176   | 75.205  |
| inter5shuttle| 207.323| 137.211| 144.747| 144.125| 137.596| 145.491| 144.957| 120.003 | 143.4   |
| inter6shuttle| 253.166| 164.355| 181.19 | 177.138| 164.924| 182.664| 178.474| 133.288 | 181.967 |
| nom10bike_day | 4.515 | 42.074 | 33.725 | 71.745 | 31.505 | 24.71  | 52.249 | 7.099   | 26.318  |
| nom10crx    | 1.227  | 3.105  | 5.409  | 7.776  | 2.828  | 4.792  | 6.855  | 0.944   | 6.939   |
| nom10magic  | 486.926| 1503.38 | 977.612| 1547.33| 1322.62| 790.61 | 1246.06| 206.797 | 821.269 |
| nom10shuttle| 1.455  | 1.14   | 1.19   | 1.234  | 1.102  | 1.134  | 1.166  | 0.425   | 0.53    |
| nom15magic  | 3358.44| 10.499.8| 6442.54| 14.838.1| 8620.79| 5060.17| 11.277 | 1509.86 | 5363.77 |
| nom20magic  | 7882.15| 32.600.2| 16.779.1| 46.609.8| 23.129.5| 10.754.4| 33.369.5| 4437.05 | 17.424  |
| nom5bike_day | 2.58  | 13.064 | 11.32  | 17.572 | 10.855 | 9.383  | 14.517 | 2.219   | 9.251   |
| nom5bike_hour | 1893.33| 8083.01| 8412.02| 8402.16| 7248.4 | 7055.42| 7163.17| 1410.11 | 8098.72 |
| nom5crx     | 0.406  | 0.623  | 1.054  | 1.061  | 0.592  | 0.983  | 0.988  | 0.193   | 1.110   |
| nom5keg     | *      | 7707.54| 16.584.8| 13.154.5| 7564.71| 16.590.3| 13.184.1| 1936.7  | 15.305  |
| nom5shuttle | 0.693  | 0.493  | 0.511  | 0.511  | 0.481  | 0.497  | 0.5    | 0.309   | 0.320   |
| ord10bike_day | 21.884| 92.944 | 402.8  | 154.541| 90.973 | 385.489| 148.472| 24.997  | 451     |
| ord10crx    | 28.367 | 85.67  | 325.608| 93.936 | 85.735 | 325.742| 94.394 | 11.653  | 342.858 |
| ord10shuttle| 51.839 | 40.338 | 42.438 | 41.475 | 40.426 | 42.419 | 41.549 | 34.293  | 40.155  |
| ord5bike_day | 2.08  | 4.688  | 12.498 | 7.34   | 4.412  | 11.501 | 6.812  | 0.936   | 12.544  |
| ord5bike_hour | 1107.57| 1749.29| 5621.96| 2304.73| 1672.93| 5173.36| 2169.43| 321.147 | 5694.64 |
| ord5crx     | 1.468  | 2.7    | 6.696  | 3.071  | 2.701  | 6.68   | 3.062  | 0.61    | 6.957   |
| ord5magic   | 99.92  | 93.845 | 108.648| 94.28  | 93.93  | 108.733| 94.437 | 46.982  | 71.721  |
| ord5shuttle | 1.676  | 1.382  | 1.408  | 1.41   | 1.38   | 1.403  | 1.404  | 1.319   | 1.417   |
| ord6magic   | 345.392| 335.947| 447.37 | 337.462| 336.4  | 447.353| 338.321| 158.227 | 277.617 |
4.3. Evaluation

Based on the experimental evaluation in Section 4, we conclude that LinCbO is the fastest algorithm for computation of the Duquenne-Guigues basis. In some cases, it is outperformed by the attribute incremental approach. However, the attribute incremental approach seems to have enormous memory requirements as it run out of memory for several datasets.

Originally, we believed that CbO itself can make the computation faster. This motivation came from the paper by Outrata & Vychodil [22], where CbO is shown to be significantly faster than NextClosure when computing intents (see Table 5). The main reason for the speed-up is the fact that CbO uses set intersection to efficiently obtain extents during the tree descent. This feature cannot be exploited for computation of the Duquenne-Guigues basis. The CbO itself rarely seems to have a significant effect on the runtime – this was the case for datasets nom10shutle and nom5shutle. Sometimes, it lead to worse performance, for example for datasets inter10crx, inter10shuttle, and nom20magic.

However, the introduction of the reuse of attribute counters significantly improves the runtime for most datasets (see Fig. 3).

5. Conclusions and further research

The algorithm LinClosure has been considered to be slow and even worse than the naïve closure [21, 6]. In an experimental evaluation, we have shown that it can perform very fast when it can reuse its attribute counters. The reuse is enabled by using CbO.

As our future research, we want to further develop the present algorithm.

• One of the benefits of CbO is that it can be improved to avoid some unnecessary closure computations. This improvement, called pruning, is in various ways utilized in FCbO [22] and In-Close ver. 3 and higher.

Table 5: Runtimes of formal concept enumeration by NextClosure and CbO in seconds for selected datasets (source: [22])

| dataset          | mushroom     | anonymous web | adult     | internet ads |
|------------------|--------------|---------------|-----------|--------------|
| size             | 8124 x 119   | 32,711 x 296  | 48,842 x 104 | 3279 x 1557 |
| fill ratio       | 19.33 %      | 1.02 %        | 8.65 %    | 0.88 %       |
| #concepts        | 238,710      | 129,009       | 180,115   | 9192         |
| NextClosure      | 53.891       | 243.325       | 134.954   | 114.493      |
| CbO              | 0.508        | 0.238         | 0.302     | 0.332        |

21
Figure 3: Comparison of NextClosure with LinClosure with an early stop (NC$^+2$, LinCbO1, and LinCbO) for datasets in batch 2; runtimes in milliseconds on a logarithmic scale (values are from Table 4).
In the case of the Duquenne-Guigues basis, the computation of closure is much more time consuming than in the case of intents. Therefore, it seems to be a good idea to apply pruning techniques in our algorithm. Our preliminary results indicate a possible 20% speed-up.

- Generalization of LinClosure is used to compute models in generalized settings, like fuzzy attribute implications [8, 9, 10] and temporal attribute implications [23]. We will explore potential uses of LinCbO in these generalizations.

- Some attribute implications are given by the character of the data. Consider the following example. If attributes \( i_1 \) and \( i_2 \) emerge from the same multi-valued property using nominal scaling, no two objects can have both of the attributes. Therefore, the attribute implication \( \{i_1, i_2\} \Rightarrow Y \) is valid. Moreover, \( \{i_1, i_2\} \) is a pseudo-intent. This raises a natural question: When we have such knowledge about the data, can we use it to make the computation faster? Adding background knowledge in the computation of the Duquenne-Guigues basis was investigated by Kriegel [16]. We will explore this possibility for LinCbO.

- The implementation used for experimental evaluation was made to be at a similar level to the Bazhanov and Obiedkov implementations [6]. We will deliver an optimized implementation of LinCbO, possibly with a pruning technique.

Acknowledgment
The authors acknowledge support by the grants

- IGA UP 2020 of Palacký University Olomouc, No. IGA_Prf_2020_019,
- JG 2019 of Palacký University Olomouc, No. JG_2019_008.

References

[1] S. Andrews, In-Close, a fast algorithm for computing formal concepts, in: International Conference on Conceptual Structures, Springer, 2009.

[2] S. Andrews, In-Close2, a high performance formal concept miner, in: Proceedings of the 19th International Conference on Conceptual Structures for Discovering Knowledge, ICCS’11, Berlin, Heidelberg, Springer-Verlag, 2011, pp. 50–62.
[3] S. Andrews, A ‘best-of-breed’ approach for designing a fast algorithm for computing fixpoints of Galois connections, Inf. Sci. 295 (2015) 633–649.

[4] S. Andrews, Making use of empty intersections to improve the performance of ChO-type algorithms, in: International Conference on Formal Concept Analysis, Springer, 2017, pp. 56–71.

[5] S. Andrews, A new method for inheriting canonicity test failures in Close-by-One type algorithms, in: Proceedings of the Fourteenth International Conference on Concept Lattices and Their Applications, 2018, pp. 255–266.

[6] K. Bazhanov and S. A. Obiedkov, Optimizations in computing the Duquenne-Guigues basis of implications, Ann. Math. Artif. Intell. 70 (1-2) (2014) 5–24.

[7] C. Beeri and P. A. Bernstein, Computational problems related to the design of normal form relational schemas. ACM Trans. Database Syst. 4 (1) (1979) 30–59.

[8] R. Belohlavek, V. Vychodil, Graded LinClosure and its role in relational data analysis, in: Proceedings of the Fourth International Conference on Concept Lattices and Their Applications, 2006, pp. 139–154.

[9] R. Belohlavek, V. Vychodil, Attribute dependencies for data with grades I, Int. J. Gen. Syst. 45 (7-8) (2016) 864–888.

[10] R. Belohlavek, V. Vychodil, Attribute dependencies for data with grades II, Int. J. Gen. Syst. 46 (1) (2017) 66–92.

[11] C. Carpineto, G. Romano, Exploiting the potential of concept lattices for information retrieval with CREDO, J. UCS 10 (8) (2004) 985–1013.

[12] D. Dua, C. Graff, UCI Machine Learning Repository, 2017.

[13] B. Ganter, R. Wille, Formal Concept Analysis – Mathematical Foundations, Springer, 1999.

[14] B. Ganter, K. Reuter, Finding all closed sets: A general approach, Order 8 (3) (1991) 283–290.

[15] J. L. Guigues, V. Duquenne, Familles minimales d’implications informatives resultant d’un tableau de données binaires, Math. Sci. Humaines 95 (1986) 5–18.
[16] F. Kriegel, D. Borchmann, NextClosures: parallel computation of the canonical base with background knowledge, Int. J. Gen. Syst. 46 (5) (2017) 490–510.

[17] S. O. Kuznetsov, A fast algorithm for computing all intersections of objects from an arbitrary semilattice, Nauchno-Tekhnicheskaya Informatsiya Seriya 2-Informatsionnye Protsessy i Sistemy (1) (1993) 17–20.

[18] S. O. Kuznetsov, On the intractability of computing the Duquenne-Guigues base, J. UCS 10 (8) (2004) 927–933.

[19] S. O. Kuznetsov, S. Obiedkov, Comparing performance of algorithms for generating concept lattices, J. Exp. Theor. Artif. Intell. 14 (2002) 189–216.

[20] D. Maier, The theory of relational databases, volume 11, Computer science press Rockville, 1983.

[21] S. Obiedkov, V. Duquenne, Attribute-incremental construction of the canonical implication basis, Ann. Math. Artif. Intell. 49 (1-4) (2007) 77–99.

[22] J. Outrata, V. Vychodil, Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data, Inf. Sci. 185 (1) (2012) 114–127.

[23] J. Triska, V. Vychodil, Minimal bases of temporal attribute implications, Ann. Math. Artif. Intell. 83 (1) (2018) 73–97.

[24] M. Wild, Computations with finite closure systems and implications, in: International Computing and Combinatorics Conference, Springer, 1995, pp. 111–120.