Decomposing Finite Closure Operators by Attribute Exploration

Daniel Borchmann

TU Dresden, Institut für Algebra
daniel.borchmann@mailbox.tu-dresden.de
http://www.math.tu-dresden.de/~borch/

Abstract. There are examples of algorithms which allow the efficient computation of the concepts of a given formal context. However, when one wants to apply those algorithms to an arbitrary closure operator, which might not be given by a formal context, those algorithms cannot be applied directly. Therefore we want to discuss the problem of finding an appropriate formal context for a given closure operator, and present an application of attribute exploration to achieve this goal.

The occurrence of high performance algorithms for the fast computation of concepts of a formal context (as [1,8,9]) makes it possible to deal with large amounts of data (if one is interested in its formal concepts). These algorithms are mostly variants of Kuznetsov's well-known Close-by-One algorithm (as for example described in [6]). In contrast to Ganter's NextClosure [3], which computes the fixpoints of a given closure operator, Close-by-One computes the concepts of a given formal context. But still both algorithms get a closure operator as input and compute it's fixpoints. The only difference is in the representation of the closure operator c: Close-by-One requires it to be given as a formal context, whereas NextClosure does not place any restrictions on the representation of c.
We want to take this asymmetry of representing a closure operator as motivation to consider the following question: Can we effectively (and also maybe efficiently) compute for a given closure operator c on a set M a formal context whose intents are *precisely* the closed sets of c? We shall formalise this idea as *decomposing closure operators* and shall show a neat application of attribute exploration in solving this problem. We also shall discuss some complexity aspects and show some experimental evaluation.

1 Formal Concept Analysis and Closure Operators

To better understand the following discussion, we briefly introduce the necessary definitions from Formal Concept Analysis and the theory of closure operators.
The basic structure used by Formal Concept Analysis is the one of a *formal context*. Let G and M be two sets and let $I \subseteq G \times M$. We will call the triple (G, M, I) a formal context. Intuitively, the set G is the set of *objects*, the set M

is the set of *attributes* and I is an *incidence relation*, where for an object $g \in G$ and an attribute $m \in M$, $g\ I\ m$ if and only if "object g has attribute m".
Now let $A \subseteq G$ and $B \subseteq M$. We define the *derivation in* (G, M, I) by

$$A' := \{\, m \in M \mid \forall g \in A : g\ I\ m \,\}$$
$$B' := \{\, g \in G \mid \forall m \in B : g\ I\ m \,\}.$$

We call the pair (A, B) a *formal concept of the formal context* (G, M, I) if and only if $A' = B$ and $B' = A$. In this case, the set A is called the *extent* and the set B is called the *intent of the formal concept* (A, B). We denote with $\mathfrak{B}(\mathbb{K})$ the set of all formal concepts of the formal context $\mathbb{K} = (G, M, I)$ and with $\mathrm{Int}(\mathbb{K})$ the set of all intents of the formal context $\mathbb{K}$.
We call a formal context $\mathbb{K} = (G, M, I)$ *object-clarified*, if for every two objects $g, h \in G$ with $\{\, g \,\}' = \{\, h \,\}'$ we already have $g = h$. That is, $\mathbb{K}$ is object-clarified if no two objects with the same set of common attributes exist in $\mathbb{K}$. Furthermore, we call an object $g \in G$ *reducible* if and only if the formal context $\tilde{\mathbb{K}}$ obtained from $\mathbb{K}$ by removing g has the same set of intents as $\mathbb{K}$. It is well known [4] that for finite formal contexts $\mathbb{K}$, i.e. where the sets G and M are both finite, that an object $g \in G$ is reducible if and only if there exists $H \subseteq G \setminus \{\, g \,\}$ such that

$$\{\, g \,\}' = \bigcap_{h \in H} \{\, h \,\}'.$$

Finally, a finite formal context $\mathbb{K}$ is called *object-reduced* if it is object-clarified and does not contain any reducible objects.
For a formal context $\mathbb{K} = (G, M, I)$ and two sets $A, B \subseteq M$ we will call the pair (A, B) an *implication of* $\mathbb{K}$ and write $A \to B$. The implication $A \to B$ is *valid in* $\mathbb{K}$ (or *holds in* $\mathbb{K}$) if and only if $B \subseteq A''$. For a set $\mathcal{L}$ of implications of $\mathbb{K}$ and a set $C \subseteq M$ we say that C is *closed under all implications from* $\mathcal{L}$ if for every implication $A \to B \in \mathcal{L}$ we have $A \not\subseteq C$ or $B \subseteq C$. That is, whenever $A \subseteq C$ then $B \subseteq C$ as well. We denote with $\mathcal{L}(C)$ the smallest set containing C that is closed under all implications from $\mathcal{L}$. This set always exists and can be computed in time linear in $|M|$ and $|\mathcal{L}|$.
Let $\mathcal{L}$ be a set of implications of a given formal context $\mathbb{K}$. We say that $\mathcal{L}$ is *sound for* $\mathbb{K}$ if all implications in $\mathcal{L}$ are valid in $\mathbb{K}$. We say that $\mathcal{L}$ is *complete for* $\mathbb{K}$ if every valid implication of $\mathbb{K}$ follows from $\mathcal{L}$. Thereby an implication $A \to B$ of $\mathbb{K}$ *follows* from $\mathcal{L}$ if and only if $B \subseteq \mathcal{L}(A)$. For every sound and complete set $\mathcal{L}$ of implications of $\mathbb{K}$ it is well known that

$$N'' = \mathcal{L}(N)$$

for all $N \subseteq M$. Again, see [4] for details.
A famous algorithm we want to utilise in this paper is the one of *attribute exploration*. This algorithm gets a formal context $\mathbb{K}$ as input and successively generates implications $A \to B$ of $\mathbb{K}$. It then asks an external expert whether the generated implication $A \to B$ should hold in $\mathbb{K}$. If it does, $A \to B$ is added to the set of valid implications. If it does not, the expert is asked for a counterexample

to falsify this implication. Thereby a counterexample is a new object g together with its attributes N such that

$$A \subseteq N \text{ and } B \not\subseteq N.$$

The formal context $\mathbb{K}$ is then extended by this new object and its attributes and the algorithm continues. It stops if no more implications can be generated.
Attribute exploration has some remarkable properties. The one we want to use in this paper is that, after the exploration has finished, we have constructed a formal context $\mathbb{K}$ and a set $\mathcal{L}$ of implications of $\mathbb{K}$, such that the set $\mathcal{L}$ is a sound and complete set of implications for $\mathbb{K}$. Additionally, $\mathcal{L}$ has minimal cardinality among all sound and complete sets for $\mathbb{K}$.
Besides the basic notions of Formal Concept Analysis, we briefly mention closure operators. A *closure operator c on a set M* is a mapping $c : \mathfrak{P}(M) \to \mathfrak{P}(M)$ such that

- $A \subseteq c(A)$ for every $A \subseteq M$,
- $A \subseteq B \implies c(A) \subseteq c(B)$ for every $A, B \subseteq M$ and
- $c(A) = c(c(A))$ for every $A \subseteq M$.

A set $N \subseteq M$ is *closed under c* if and only if $c(N) = N$. We call c a *finite* closure operator if M is a finite set.
Finally, we briefly mention the definition of meet-irreducible closed sets. Let c be a closure operator on a set M. Then a closed set $N \in c[\mathfrak{P}(M)]$ is called *meet-irreducible (in $(c[\mathfrak{P}(M)], \subseteq)$)* if and only if

$$N \neq \bigcap \{\, \tilde{N} \in c[\mathfrak{P}(M)] \mid \tilde{N} \supsetneq N \,\},$$

or equivalently, N cannot be represented as an intersection of strictly greater closed sets of c.

2 Problem Specification

Now let c be a closure operator on M for some set M. To somehow represent c by a formal context $\mathbb{K}$ we demand that its intents are exactly the fixpoints of c.

Definition 1 Let c be a closure operator on a set M. We say that a formal context $\mathbb{K} = (G, M, I)$ is a *(contextual) decomposition of c* if

$$\mathrm{Int}(\mathbb{K}) = c[\mathfrak{P}(M)]. \qquad \diamondsuit$$

Note that this means that for a decomposition of a closure operator c into the formal context $\mathbb{K} = (G, M, I)$ it holds that

$$c(N) = N''$$

for all $N \subseteq M$. Note that this condition is also sufficient for $\mathbb{K}$ being a decomposition of c.

The equality $c = \cdot''$ also motivates the notion of *decomposition*, since we can now represent the closure operator c as the composition of the two mappings $\cdot' : \mathfrak{P}(M) \to \mathfrak{P}(G)$ and $\cdot' : \mathfrak{P}(G) \to \mathfrak{P}(M)$.
In theory, the problem of finding a decomposition of a given closure operator has an easy solution, as for every closure operator c we can explicitly give a decomposition of c.

Proposition 2 ([4]) *Let c be a closure operator on a set M. Then the formal context*

$$\mathbb{K}_c = (c[\mathfrak{P}(M)], M, \ni)$$

is a decomposition of c.

Proof We show that for every set $N \subseteq M$ the equality $c(N) = N''$ holds in $\mathbb{K}_c$. For this we observe that

$$N' = \{\, c(A) \mid A \subseteq M, N \subseteq c(A) \,\}.$$

This yields

$$\begin{aligned} N'' &= \{\, x \in M \mid \forall A \in N' : x \in A \,\} \\ &= \bigcap N' \\ &= \bigcap \{\, c(A) \mid A \subseteq M, N \subseteq c(A) \,\} \\ &= c(N) \end{aligned}$$

as required. □

Unfortunately, this solution is rather impractical, since the size of $\mathbb{K}_c$ is correlated to the number of closed sets of c, which is often exponential in the size of M. Luckily, an object-reduced subcontext of $\mathbb{K}_c$ can be given explicitly by removing all non-meet-irreducible closures from the objects of $\mathbb{K}_c$.

Proposition 3 *Let c be a closure operator on a finite set M and let $C \subseteq c[\mathfrak{P}(M)]$ be the set of meet-irreducible sets in $(c[\mathfrak{P}(M)], \subseteq)$. Then the formal context*

$$\mathbb{K} = (C, M, \ni)$$

is an object-reduced decomposition of c.

Proof Recall that an object in $\mathbb{K}_c$ is reducible if its attributes can be represented as an intersection of attributes of other objects. This immediately gives that a closed set is reducible in $\mathbb{K}_c$ if it is not meet-irreducible. Therefore $\mathbb{K}$ is the object-reduced subcontext of $\mathbb{K}_c$ and $\operatorname{Int}(\mathbb{K}) = \operatorname{Int}(\mathbb{K}_c) = c[\mathfrak{P}(M)]$ as required.□

Note that the above proposition is not true for infinite sets M. For this consider the identity on the set $\mathbb{N}$ of natural numbers, which is a (trivial) closure operator. Then $(\mathfrak{P}(\mathbb{N}), \subseteq)$ does not have meet-irreducible sets at all, but the formal context $(\emptyset, \mathbb{N}, \emptyset)$ is not a decomposition of the identity on $\mathbb{N}$.

Note that, unfortunately (again), there is up to now no easy way to directly compute those meet-irreducible closures and thus no easy way to directly compute this object-reduced subcontext.
The formal context $\mathbb{K}_c$ is a very universal decomposition of the finite closure operator c, in the following sense: Whenever we have a formal context $\mathbb{K}$ which is a decomposition of c, we can find a subcontext of $\mathbb{K}_c$ which is equal to $\mathbb{K}$ up to object renaming. This observation is captured in the following result.

Proposition 4 *Let c be a finite closure operator on a set M and let $\mathbb{K} = (G, M, I)$ be an object-clarified decomposition of c. Let $\mathbb{K}_c = (H, M, J)$. Then there exists a set $N \subseteq H$ and a bijective mapping $\alpha : G \to N$ such that*

$$g\ I\ m \iff \alpha(g)\ J\ m$$

for all $g \in G$.

Proof Let $g \in G$. Then $\{ g \}'$ is an intent of $\mathbb{K}$ and therefore $\{ g \}'$ is closed with respect to c, i.e. $c(\{ g \}') = \{ g \}'$. Therefore, there exists an element $h \in H$ such that $\{ g \}' = \{ h \}'$, whereby the first derivation is done in $\mathbb{K}$ and the second is done in $\mathbb{K}_c$. Now set $\alpha(g) := h$. Then α is a bijective mapping from G to $N := \alpha[G]$. □

Since we know that for a given finite formal context $\mathbb{K}$ its object-reduced subcontext is uniquely determined, we immediately have the following result.

Proposition 5 *For every finite closure operator c all its object-reduced decompositions are equal up to object renaming.*

3 Using Attribute Exploration to Decompose Closure Operators

So, let us reconsider the problem statement: We have given the set of attributes of the context $\mathbb{K}$ we are looking for. Additionally, we know whether an implication in $\mathbb{K}$ holds or not. This is the case because with our closure operator c, we can decide whether $A \to B$ should hold in a decomposition of c by checking

$$B \subseteq c(A).$$

If this is not true, then $c(A)$ is a set of attributes which is a counterexample to the implication $A \to B$ (since we know that $c(A)$ will be an intent of the final context).
What we now see is that we can use c as the *expert during attribute exploration* which is able to automatically verify whether an implication holds or not, and in the case it doesn't can automatically provide a counterexample.
This immediately leads us to the pseudo code shown in Algorithm 1. It works as follows:

Algorithm 1. Decomposing closure operators using attribute exploration

```
define context-for-closure-operator(M, c)
  let (𝕂 = explore-attributes((∅, M, ∅),
                              function (A → B)
                                if B ⊆ c(A) then
                                  return holds
                                else
                                  return fails, (x, c(A))
                                end if
                              end function))
    return object-reduce(𝕂)
  end let
end define
```

– Given a set M and a closure operator c on M we call `explore-attributes`, which gets as arguments a starting context (which is $(\emptyset, M, \emptyset)$) and a function to call with a possible implication $A \to B$. This function returns either *holds*, if $A \to B$ holds. Otherwise it returns *fails* together with a counterexample of the form (x, N). Here x is a new object not already present in the current formal context. The set N is the set of attributes the new object x should have. This means that we add x as a new object and extend the incidence relation by the set

$$\{\, (x, a) \mid a \in N \,\}.$$

When `explore-attributes` has finished, it returns the context it has constructed during exploration. This is the context $\mathbb{K}$.
– Then we remove redundant objects from $\mathbb{K}$ and return the result.

Proposition 6 *The function* `context-for-closure-operator` *computes for a given finite closure operator c on a given set M the object-reduced decomposition of c.*

Proof Denote with $\mathbb{K} = (\mathcal{G}, M, I)$ the formal context computed by the call to `explore-attributes`. It is clear that $\mathbb{K}$ is object-clarified, since in any iteration, if $A \to B$ does not hold, the set $c(A)$ is a counterexample, but no previously given set is. Therefore $c(A)$ cannot occur more then once and therefore there are no two objects in $\mathbb{K}$ with the same set of common attributes.
We show $c(N) = N''$ for every set $N \subseteq M$. Since $\mathcal{G} \subseteq c[\mathfrak{P}(M)]$ the following holds

$$\begin{aligned} N'' &= \bigcap \{\, c(A) \in \mathcal{G} \mid N \subseteq c(A) \,\} \\ &\supseteq \bigcap \{\, c(A) \in c[\mathfrak{P}(M)] \mid N \subseteq c(A) \,\} \\ &= c(N). \end{aligned}$$

Conversely let $\mathcal{L}$ be the set of implications accepted during the attribute exploration. We then know that $\mathcal{L}$ is sound and complete for $\mathbb{K}$ and therefore $N'' = \mathcal{L}(N)$ for every $N \subseteq M$. But for every implication $A \to B \in \mathcal{L}$ holds that $B \subseteq c(A)$. Therefore, $c(N)$ is closed under all implications from $\mathcal{L}$, since $A \subseteq c(N)$ implies $B \subseteq c(A) \subseteq c(N)$. Furthermore, $c(N) \supseteq N$. But N'' is the smallest set that contains N and is closed under all implications from $\mathcal{L}$. Thus $c(N) \supseteq N''$.
In sum, we have $c(N) = N''$ for every $N \subseteq M$, hence the formal context $\mathbb{K}$ computed by `explore-attributes` is a decomposition of c. After removing all redundant objects from $\mathbb{K}$, we get an object-reduced decomposition of c, as required. □

Clearly, $c(A)$ is a counterexample if $A \to B$ does not hold, but it is not the only one. Instead of simply giving $c(A)$ as a counterexample it would be much better to search for a "stronger" counterexample, which also falsifies $A \to B$, but provides even some more information. This is the topic of the next section.

4 Maximal Counterexamples

We can tell a bit more about the objects in the object-reduced subcontext mentioned in Section 2. The meet-irreducible closures can be characterised as follows.

Proposition 7 *Let c be a closure operator on a set M. Then a set $N \in c[\mathfrak{P}(M)]$ is meet-irreducible in $(c[\mathfrak{P}(M)], \subseteq)$ if and only if there exists $n \in M$ with $n \notin N$ and N is maximal among all closed sets of c with this property.*

Proof Let $N \in c[\mathfrak{P}(M)]$ be meet-irreducible in $(c[\mathfrak{P}(M)], \subseteq)$ and let $\mathcal{N} = \{\, \tilde{N} \in c[\mathfrak{P}(M)] \mid \tilde{N} \supsetneq N \,\}$. Then $\bigcap \mathcal{N} \neq N$, i.e. $\bigcap \mathcal{N} \setminus N \neq \emptyset$. Fix one $n \in \bigcap \mathcal{N} \setminus N$. Then $n \notin N$ and $n \in \tilde{N}$ for every $\tilde{N} \supsetneq N$ with $\tilde{N} \in c[\mathfrak{P}(M)]$, as required.
Conversely, let $n \notin N$ such that $n \in \tilde{N}$ for every $\tilde{N} \supsetneq N$ with $\tilde{N} \in c[\mathfrak{P}(M)]$. Then

$$n \in \bigcap \{\, \tilde{N} \in c[\mathfrak{P}(M)] \mid \tilde{N} \supseteq N \,\}$$

and since $n \notin N$ if follows that

$$N \neq \bigcap \{\, \tilde{N} \in c[\mathfrak{P}(M)] \mid \tilde{N} \supseteq N \,\}.$$ □

With this proposition in mind it is easy to see what better counterexample for $A \to B$ we can give as simply returning $c(A)$. If we have found that $B \not\subseteq c(A)$ we choose $n \in B \setminus c(A)$ and look for a set $C \supseteq c(A)$ with $n \notin C$ which is maximal with this property. This can be done in time $\mathcal{O}(|M|^2)$, as the pseudo code in Algorithm 2 shows.
However, we can do better than $\mathcal{O}(|M|^2)$. For this we observe that an element $m \in R$ such that $n \in c(C \cup \{\, m \,\})$ has not to be considered again. To see this, consider an iteration of the main loop of Algorithm 2 and fix the current value of C as $\bar{C}$ and fix $m \in R$ such that $n \in c(C \cup \{\, m \,\})$. For any subsequent iteration

Algorithm 2. Computing maximal counterexamples

```
define maximal-counterexample(M, c, A → B)
  let (n = some element in B \ c(A))
    loop (C = c(A),
          R = M \ {n} \ c(A))
      if there exists some m ∈ R such that n ∉ c(C ∪ {m}) then
        recur with (C = c(C ∪ {m}), R = R \ {m})
      else
        return C
      end if
    end loop
  end let
end define
```

let $\tilde{C}$ be the value of C of Algorithm 2 in that iteration. Then we know that $\bar{C} \cup \{m\} \subseteq \tilde{C} \cup \{m\}$ and since $n \in c(\bar{C} \cup \{m\}) \subseteq c(\tilde{C} \cup \{m\})$, we do not have to consider the element m again. This leads us to the refined version of computing maximal counterexamples as shown in Algorithm 3.
The following result is now apparent.

Proposition 8 *The function* `maximal-counterexample` *as shown in Algorithm 3 computes for a given finite closure operator c on a given set M a meet-irreducible counterexample N for $A \to B$, i.e. $A \subseteq N$, $B \not\subseteq N$ and N is a meet-irreducible closed set of c.*

Now the function given as the second argument to `explore-attributes` in Algorithm 1 should return the new maximal counterexample instead of $c(A)$. Given this modification we can make the following observation for an arbitrary closure operator c: Since the created counterexamples are objects of an object-reduced subcontext of $\mathbb{K}_c$ they cannot be reducible in the final context returned from the attribute exploration. This yields a new implementation of `context-for-closure-operator` as shown in Algorithm 4.

Proposition 9 *The function* `context-for-closure-operator` *as shown in Algorithm 4 computes for every finite closure operator c on a set M an object-reduced decomposition of c.*

Proof Denote with $\mathbb{K}$ the context computed by `explore-attributes`. By the same argument as in Proposition 6 we see that $\mathbb{K}$ is an object-clarified decomposition of c. It remains to show that $\mathbb{K}$ is object-reduced. But this is clear since the only objects added during the attribute exploration are meet-irreducible closed sets. □

Algorithm 3. Computing maximal counterexamples in time $\mathcal{O}(|M|)$.

```
define maximal-counterexample(M, c, A → B)
  let (n = some element in B \ c(A))
    loop (C = c(A),
          R = M \ {n} \ c(A))
      if R = ∅ then
        return C
      else
        let (m be some element in R)
          if n ∉ c(C ∪ {m}) then
            recur with (C = c(C ∪ {m}), R = R \ {m})
          else
            recur with (C = C, R = R \ {m})
          end if
        end let
      end if
    end loop
  end let
end define
```

Algorithm 4. Decomposing closure operators with attribute exploration

```
define context-for-closure-operator(M, c)
  return explore-attributes(
           (∅, M, ∅),
           function (A → B)
             if B ⊆ c(A) then
               return holds
             else
               return fails, (x, maximal-counterexample(M, c, A → B))
             end if
           end function)
end define
```

5 Complexity Considerations

We have seen that we can use attribute exploration to directly compute an object-reduced decomposition of a given finite closure operator c on a set M. The question remains, however, if this approach really is better then the naive computation of the formal context $\mathbb{K}_c$ and reducing it afterwards.

Firstly we observe that the size of the object-reduced decomposition of c can be exponential in $|M|$. For this we recall that a relation $R \subseteq M \times M$ on a set M is an *equivalence relation* if R is symmetric, transitive and reflexive. For a given set $P \subseteq M \times M$ we denote with $c(P)$ the smallest equivalence relation that contains all elements of P. Then c is a finite closure operator on $M \times M$. Now the meet-irreducible equivalence relations are precisely those that are maximal with respect to not containing a given pair $(x, y) \in M \times M$ with $x \neq y$. For each such pair (x, y) we can choose a set $S \subseteq M$ such that $x \in S, y \notin S$. Then the set $\{ S, M \setminus S \}$ is a partition of such a meet-irreducible equivalence relation and every such partition uniquely induces a meet-irreducible equivalence relation. We call such a partition a *split on* M (c.f. [4]) and easily observe that there are $2^{|M|-1} - 1$ such splits on M. Thus c has exponentially many meet-irreducible closed sets and therefore every decomposition of c will be exponentially large in $|M \times M|$.

Thus we cannot hope that Algorithm 4 can compute a decomposition of c in time polynomial in $|M|$. However, we can ask whether we can compute a decomposition of c in time polynomial in the *output*, i.e. in the size of the decomposition itself. The time needed to compute the decomposition as shown in Algorithm 4 corresponds to the number of calls to the expert during attribute exploration, and to the time needed to compute the next implication. Unfortunately, it is known that the number the expert is called can be exponential, since the minimal size of a sound and complete set of implications can be exponential in the size of the formal context [5]. Unfortunately (again), it is also well known that the time between the computation of two implications can also be exponential in the size of the formal context [2]. Therefore our algorithm cannot compute a decomposition of c in time polynomial in the size of the decomposition.

Another aspect we have not considered so far is the following. Up to now we only asked for the complexity with respect to the size of M, but never with respect to the size of c. This is because it is not clear in which way the closure operator c should be encoded. If we would simply encode c as a table of argument-value pairs, then the input would be exponentially large in $|M|$, and complexity considerations would be trivial, since everything can be done in time polynomial in the input. Other encodings may yield other results, but this is not clear by now, and we will discuss this shortly in Section 7.

6 Experimental Evaluation

Since theoretical complexity considerations are out of the scope of this paper, we at least want to give some experimental results.

For this, we have implemented three different algorithms to compute the object-reduced decomposition of a given finite closure operator c. Those are

A) The direct computation of $\mathbb{K}_c$ with a subsequent object-reduction, using NextClosure to compute all intents,
B) Algorithm 1,
C) Algorithm 4,

For our experiments we want to consider closure operators on an 11-elemental set. To randomly (but not uniformly randomly) generate closure operators over this set we simply generate formal contexts $\mathbb{K} = (G, M, I)$ with $M = \{0, \ldots, 10\}$ and some arbitrary $G \subseteq \mathfrak{P}(M)$ and $I \subseteq G \times M$. Our closure operator then will be $\cdot'' : \mathfrak{P}(M) \to \mathfrak{P}(M)$. The randomly generated formal contexts have been reduced before the experiments where conducted.
When examining the performance of the algorithms given above, we are not only interested in the time they take to accomplish their task. We are also interested in the number of times the closure operator is invoked. As it turns out, this number somehow correlates to the number of concepts of $\mathbb{K}$, which is equal to the number of closed sets of $\cdot''$, or it is correlated to the number of *pseudo-intents* of $\mathbb{K}$.
Pseudo-intents play an important role in attribute exploration, as any asked implications $A \to B$ is such that the set A is a pseudo-intent of the currently known formal context. It therefore seems reasonable that they will have a non-trivial impact on the performance of the algorithms using attribute exploration. In the following, we want to examine this conjuncture.
But before we do so, let us consider the actual speedups of our new algorithms. For this we show in Figure 1 the performance improvements of the algorithms B and C against algorithm A. All running times are given in seconds.

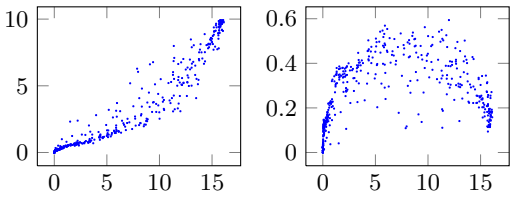

Fig. 1. Runtime of algorithm A compared to the other algorithms

As we can see, algorithm B allows for about 50% of speedup compared to algorithm A, whereas algorithm C even shows performance improvements between factors of 5 to 15, in extreme cases even factors up to 150.
What is surprising at the second picture is its very different form, compared to the first one. This is related to the number of pseudo-intents, as we already have mentioned, and as also Figure 2 might suggest. In that picture we see a correlation between the number of intents and the number of pseudo-intents of

our sample contexts. The overall appearance of this picture is very similar to the second one in Figure 1, and as we will see shortly, the performance of algorithm C is indeed dominated by the number of pseudo-intents.

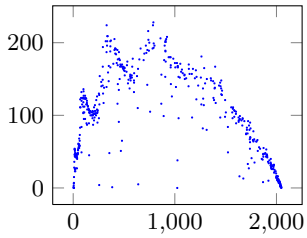

Fig. 2. Number of intents against number of pseudo-intents

Now let us examine our algorithms in more depth. For this we show in Figures 3, 4 and 5 the performance of our algorithms drawn against the number of intents, pseudo-intents and calls to the closure operator $\cdot''$. The results for algorithms A, B and C are given from left to right.
In Figure 3 we can see that the runtime of the algorithms A and B clearly depends on the number of intents. Algorithm C, on the other hand, produces a picture which is more like the one shown in Figure 2.
In contrast to this, Figure 4 reverses the roles. For algorithms A and B we can find again the picture of Figure 2, with the roles of the axes reversed. For algorithm C there seems to be a dependency between pseudo-intents and runtime, but it is only very vague. At best, one might conjecture that the points distribute around the graph of a linear function.
Finally, the last picture we want to consider is the one in Figure 5. We can see that the time grows superlinear in the number of concepts for both algorithm A and B, whereby the dependency is much clearer in the case for algorithm B. Surprisingly, it seems that the runtime depends quadratically on the number of calls to $\cdot''$ in both cases. The case for algorithm B seems even more astonishing, as the dependence seems to be *functional*, and not of statistical nature. It is not clear to the author how this result can be explained thoroughly.
Unfortunately, the picture is not that clear anymore when we consider algorithm C, and the results are quite hard to interpret. One can clearly spot a linear dependency between the number of calls to $\cdot''$ and the overall runtime. However, not all sample contexts seem to obey this linear dependency, as they seem to accumulate in other *branches*. Again, it is not clear to the author how this distribution can be explained in a mathematically thorough way.

7 Conclusions

We have presented a small application of attribute exploration for the problem of finding a decomposition of an arbitrary finite closure operator c. By using

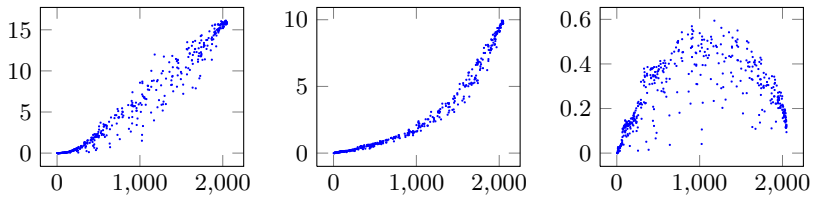

Fig. 3. Number of intents against overall runtime

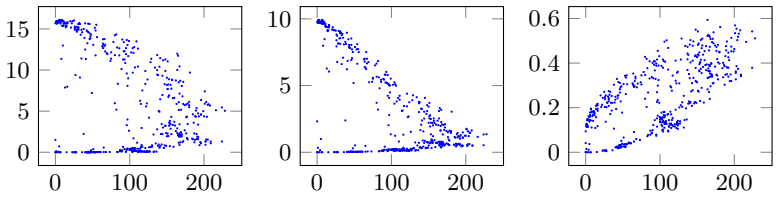

Fig. 4. Number of pseudo-intents against overall runtime

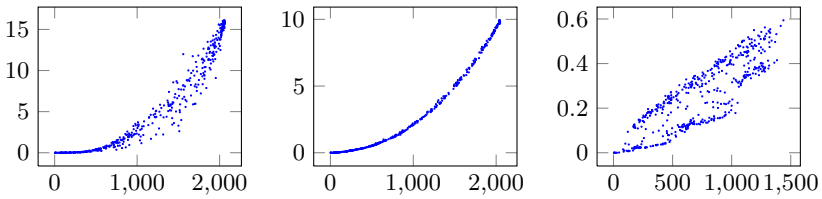

Fig. 5. Calls to $\cdot''$ against overall runtime

maximal counterexamples instead of smaller ones we were also able to show that we can directly compute an object-reduced decomposition without some intermediate, non-object-reduced context.
By some theoretical result we were able to show that our new algorithm is not able to compute the object-reduced decomposition of an arbitrary closure operator efficiently, since it may always be exponentially large in the size of the input set. However, experiments show that the new algorithms, in particular when using maximal counterexamples, were able to outperform the naive implementations. But the experiments also revealed unexpected results and a satisfactory explanation could not be given. This clearly is a point of future research. In those experiments it is also not clear in which way the generation of the sample contexts had an influence on the overall results of the experiments, especially the one shown in Figure 2, which, at least for the author, was unexpected.
Another point, which has been already mentioned above, is the following: For a thorough complexity analysis of the presented algorithm it must be clarified how to represent the closure operator c. Apart from trivially listing all possible closed sets, it might also be possible to represent c as a complete set of implications. For such a representation we can see that the object-reduced decomposition of c can then be computed in space polynomial in the input and the output. It is not clear, however, if other representations yield better complexity results.

8 Acknowledgements

The author is deeply indebted to the thorough reviews of the anonymous reviewers, which substantially improved the overall quality of this paper. The author would also like to thank Bernhard Ganter, who suggested the $\mathcal{O}(|M|)$ computation of the maximal counterexamples.

References

1. Simon Andrews. In-Close, a fast algorithm for computing formal concepts. In Sebastian Rudolph, Frithjof Dau, and Sergei O. Kuznetsov, editors, *ICCS Supplementary Proceedings*, Moscow, 2009.
2. Felix Distel. Hardness of Enumerating Pseudo-intents in the Lectic Order. In Kwuida and Sertkaya [7], pages 124–137.
3. Bernhard Ganter. Two basic algorithms in concept analysis. In Kwuida and Sertkaya [7], pages 312–340.
4. Bernhard Ganter and Rudolph Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin-Heidelberg, 1999.
5. Sergei O. Kuznetsov. On the intractability of Computing the Duquenne-Guigues Base. *J. UCS*, 10(8):927–933, 2004.
6. Sergei O. Kuznetsov and Sergei A. Obiedkov. Algorithms for the construction of concept lattices and their diagram graphs. In Luc De Raedt and Arno Siebes, editors, *PKDD*, volume 2168 of *Lecture Notes in Computer Science*, pages 289–300. Springer, 2001.

7. Léonard Kwuida and Baris Sertkaya, editors. *Formal Concept Analysis, 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings*, volume 5986 of *Lecture Notes in Computer Science*. Springer, 2010.
8. Vilém Vychodil, Petr Krajča, and Jan Outrata. Parallel Recursive Algorithm for FCA. In Radim Bělohlávek and Sergej O. Kuznetsov, editors, *Concept Lattices and Their Application*, pages 71–82. Palacký University, Olomouc, 2008.
9. Vilém Vychodil, Petr Krajča, and Jan Outrata. Advances in algorithms based on CbO. In Marzena Kryszkiewicz and Sergei Obiedkov, editors, *Concept Lattices and Their Application*, pages 325–337, 2010.