Comparing Algorithms for Computing Lower Covers of Implication-closed Sets

Alexandre Bazin

LIMOS, CNRS, Blaise Pascal University, Clermont-Ferrand, France
contact@alexandrebazin.com

Abstract. In this paper, we consider two methods for computing lower cover of elements in closure systems for which we know an implicational basis: intersecting meet-irreducible elements or computing minimal transversals of sets of minimal generators. We provide experimental results on the runtimes for single computations of lower covers and depth-first searches.

1 Introduction

Closed sets are essential objects in many fields such as data mining or database theory. Most of the time, one is interested in computing all or parts of the closure system for a given closure operator [6, 8]. However, we are sometimes faced with the problem of finding a specific closed set that respects some property and computing too much of the rest of the closure system is a waste of time. An example of this is the problem of finding a maximal frequent closed set. As frequency is an anti-monotone property, it suffices to start with the least closed set and perform a depth-first search by repeatedly computing upper covers of sets until we cannot find frequent sets anymore. Given a closure operator and a closed set, it is easy to compute the closed sets immediately above it. Problems start to arise when we are looking for specific closed sets respecting a property that is not anti-monotone. For example, when computing the Duquenne-Guigues basis $\mathcal{B}$ for some closure operator c, a pseudo-closed set P is $\mathcal{I}$-closed but not c-closed for $\mathcal{I} = \mathcal{B} \setminus \{P \rightarrow c(P)\}$. The non-anti-monotonicity of the c-closedness prevents us from using depth-first searches from the bottom.

What we are interested in is the problem of performing depth-first searches starting from the maximal element in closure systems for which we know an implicational basis. This amounts to computing lower covers of closed sets using the implications, a problem shown to be NP-hard [1]. In this work, we compare two algorithms - one based on meet-irreducibles, the other on minimal generators - that solve this problem. Section 2 reviews basic definitions and properties of lower covers that the algorithms use. Sections 3 and 4 respectively describe the methods using meet-irreducible elements and minimal generators. In Section 5, we present experimental results on the runtimes for both algorithms.

2 Preliminaries

2.1 Basic Definitions

Let us use E to denote a set of elements.

Definition 1 *A closure operator $c : 2^E \mapsto 2^E$ is an extensive ($X \subseteq c(X)$), increasing ($X \subseteq Y \Rightarrow c(X) \subseteq c(Y)$) and idempotent ($c(c(X)) = c(X)$) function.*

A set $S \in 2^E$ such that $S = c(S)$ is said to be *closed*. The intersection of two closed sets is closed. The set of all sets closed for a given closure operator c ordered by inclusion forms a lattice that will here be denoted by Φ_c.

Definition 2 *An implication on E is a pair $(A, B) \in 2^E \times 2^E$, most commonly written $A \rightarrow B$.*

Definition 3 *Let $\mathcal{I}$ be a set of implications. We denote by $\mathcal{I}(\cdot)$ the closure operator, sometimes called logical closure, that maps a set X to its smallest superset Y such that*

$$\forall A \rightarrow B \in \mathcal{I}, A \subseteq Y \Rightarrow B \subseteq Y$$

The logical closure is a closure operator. As such, for an implication set $\mathcal{I}$, we will use $\Phi_{\mathcal{I}}$ to denote the lattice of implication-closed sets ordered by inclusion.

Definition 4 *A minimal generator G of $X \in 2^E$ for a closure operator c is an inclusion-minimal set such that $c(G) = X$.*

In the remainder of this paper, we will use the term *minimal generator*, without any more details, to talk about the minimal generators for the logical closure. The set of minimal generators of a set S for an implication set $\mathcal{I}$ will be denoted $Gen_{\mathcal{I}}(S)$.

Definition 5 *Let $\mathcal{L} = (E, \leq)$ be a lattice. A meet-irreducible element is an element $e \subseteq E$ that has a single upper cover, i.e. $\{x \in E \mid x > e\}$ has a single inclusion-minimal element. We use $\mathcal{M}(\mathcal{L})$ to denote the set of meet-irreducible elements of the lattice $\mathcal{L}$.*

Any element of the lattice $\mathcal{L}$ is the infimum of a set of meet-irreducible elements.

Definition 6 *Let $\mathcal{H} = (\mathcal{E}, V)$ be a hypergraph. A minimal transversal of $\mathcal{E}$ is a set $S \subseteq V$ such that $\forall X \in \mathcal{E}$, $X \cap S \neq \emptyset$.*

We use $Tr(\mathcal{E})$ to denote the set of minimal transversals of a set of hyperedges $\mathcal{E}$.

2.2 Lower Covers

If we want to perform a depth-first search from the top in a lattice $\Phi_{\mathcal{I}}$ for which we know $\mathcal{I}$, we have to be able to compute the lower covers of a set $S \in \Phi_{\mathcal{I}}$. We present here two methods that can be used to compute them.

Proposition 1 *For any $S \in \Phi_{\mathcal{I}}$, the lower covers of S are the inclusion-maximal elements of $\{S \cap M \mid M \in \mathcal{M}(\Phi_{\mathcal{I}})$ and $S \nsubseteq M\}$.*

Proof Every element of $\Phi_{\mathcal{I}}$ is the infimum of a subset of $\mathcal{M}(\Phi_{\mathcal{I}})$. Additionally, $\Phi_{\mathcal{I}}$ being a closure system, the infimum of two sets is their intersection. As such, for any $M \in \mathcal{M}(\Phi_{\mathcal{I}})$, $S \cap M \subseteq S$ is an $\mathcal{I}$-closed set. $S \cap M$ being strictly contained in S, if it is inclusion-maximal then it is a lower cover of S. $\square$

Using Proposition 1 to compute lower covers requires that we first know the meet-irreducible elements of the lattice. In most case, they must be explicitly computed beforehand. Algorithm 1, presented in Section 3.1, does this.

Proposition 2 *For any $S \in \Phi_{\mathcal{I}}$, the lower covers of S are the sets $S \setminus T$ with $T \in Tr(Gen_{\mathcal{I}}(S))$.*

Proof Let T be a minimal transversal of $Gen_{\mathcal{I}}(S)$. For any $e \in S \setminus T$, $(S \setminus T) \cup \{e\} = S \setminus (T \setminus \{e\})$ contains a minimal generator of S because of the minimality of T. Therefore, there is no closed set between S and $S \setminus T$ and $S \setminus T$ is a lower cover of S. $\square$

Using Proposition 2 to compute the lower covers of S would require that we compute the minimal generators S first, then their minimal transversals. The algorithms we use for these problems are described in Section 4.

3 Computing with Meet-irreducible Elements

3.1 Computing Meet-irreducible Elements from an Implication Base

We use Wild's algorithm [9] to compute the set of meet-irreducible elements of the lattice $\Phi_{\mathcal{I}}$ for which we know an implication base $\mathcal{I}$. It is based on the fact that a set $S \in \Phi_{\mathcal{I}}$ is meet-irreducible if and only if it is maximal among closed sets that do not contain some element e. Let us call $max(e, \mathcal{I})$ the set of maximal elements of $\Phi_{\mathcal{I}}$ that do not contain e and $max'(e, \mathcal{I} \setminus \{I\})$ and $max''(e, \mathcal{I} \setminus \{I\})$ the subsets of $max(e, \mathcal{I} \setminus \{I\})$ for which $I = A \rightarrow B$ respectively holds and does not hold. We then have

$$max(e, \mathcal{I}) \subseteq max'(e, \mathcal{I} \setminus \{I\}) \bigcup_{a \in A} max''(e, \mathcal{I} \setminus \{I\}) * max(a, \mathcal{I} \setminus \{I\})$$

where $X * Y = \{x \cap y \mid x \in X, y \in Y\}$.

Algorithm 1 uses this property to compute the meet-irreducible elements of $\Phi_{\mathcal{I}}$ from $\mathcal{I} = \{A_1 \rightarrow B_1, ..., A_n \rightarrow B_n\}$ by computing the meet irreducibles of every lattice $\Phi_{\mathcal{I}_i}$ such that $\mathcal{I}_i = \{A_1 \rightarrow B_1, ..., A_i \rightarrow B_i\}$ with $0 \leq i \leq n$.

Algorithm 1: Meet-irreducibles

```
1  Meet-Irreducibles ($E$, $\mathcal{I}$)
   Input  : Implication set $\mathcal{I} = \{A_1 \rightarrow B_1, ..., A_n \rightarrow B_n\}$ on $E$
   Output: $\mathcal{M}(\Phi_{\mathcal{I}})$
2  foreach $e \in E$ do
3      $max(e) = \{E \setminus \{e\}\}$
4  end
5  foreach $i \in 1..n$ do
6      foreach $e \in E$ do
7          $max'(e) = \{Z \in max(e) \mid A_i \not\subseteq Z$ or $B_i \subseteq Z\}$
8          $max''(e) = max(e) \setminus max'(e)$
9          foreach $a \in A_i$ do
10             foreach $X \in max''(e)$ and $Y \in max(a)$ do
11                 $T(e) = T(e) \cup \{X \cap Y\}$
12             end
13         end
14         $max(e) = \{Z \in T(e) \mid Z$ is inclusion-maximal$\}$
15     end
16 end
17 $\mathcal{M}(\Phi_{\mathcal{I}}) = \bigcup_{e \in E} max(e)$
```

Algorithm 1 has a runtime exponential in the size of its output, which can itself be exponential in the size of the implication base.

3.2 Intersecting Sets

Once the meet-irreducible elements of the lattice are known, we can compute the lower covers of a set S by intersecting it with the meet-irreducible sets that are not supersets of S and keeping the inclusion-maximal elements. Algorithm 2 runs in time polynomial in the size of $\mathcal{M}(\Phi_{\mathcal{I}})$.

4 Computing with Transversals of Minimal Generators

4.1 Computing Minimal Generators

We compute the minimal generators of an implication-closed set with Algorithms 3 and 4 proposed in [7].

Algorithm 2: Intersection of meet-irreducible elements

```
Input  : A set $S$ and meet-irreducibles $\mathcal{M}(\Phi_{\mathcal{I}})$
Output: The lower covers of $S$
1 $C = \emptyset$
2 foreach $M \in \mathcal{M}(\Phi_{\mathcal{I}})$ do
3     $C = C \cup (S \cap M)$
4 end
5 Return $max(C)$
```

Algorithm 3: First minimal generator

```
1 MinGen $(P, \mathcal{L})$
  Input  : Implications $\mathcal{L}$ on the set $E$ and a subset $P \subseteq E$ such that $\mathcal{L}(P) = P$
  Output: A minimal generator $Q$ of $P$
2 $Q \leftarrow P$
3 foreach $m \in P$ do
4     if $\mathcal{L}(Q \setminus \{m\}) = P$ then
5         $Q \leftarrow Q \setminus \{m\}$
6     end
7 end
```

Algorithm 3, given a set P and an implication set $\mathcal{L}$, computes a first minimal generator of P. Algorithm 4 computes all the minimal generators of a set P for an implication set $\mathcal{L}$. It has time complexity $O(|\mathcal{L}| \times |\mathcal{G}| \times |\mathcal{P}| \times (|\mathcal{G}| + |\mathcal{P}|))$.

4.2 Computing Minimal Transversals

The problem of computing minimal transversals is a classic that, while extensively studied, still holds many interesting question [4, 5, 3]. It has been shown to be solvable in quasi-polynomial total time. Here, we chose to compute minimal transversals using the, arguably, most simple algorithm, *Berge Multiplication* [2]. Given two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$, their *edgewise union* is defined as :

$$\mathcal{H}_1 \vee \mathcal{H}_2 = \{h_1 \cup h_2 \mid h_1 \in \mathcal{H}_1 \text{ and } h_2 \in \mathcal{H}_2\}$$

We then have

$$Tr(\mathcal{H}_1 \cup \mathcal{H}_2 = min(Tr(\mathcal{H}_1) \vee Tr(\mathcal{H}_2))$$

Which gives rise to Algorithm 5.

5 Experimental Results

We implemented both methods, used them on randomly generated implicational bases and compared their runtimes. Implications were generated by using

Algorithm 4: All minimal generators

```
Input  : Implication set $\mathcal{L}$ on the attribute set $E$ and an $\mathcal{L}$-closed set $P \subseteq E$
Output: All minimal generators $\mathcal{G}$ of $P$
1  $\mathcal{G} \leftarrow MinGen(P, \mathcal{L})$
2  foreach $G \in \mathcal{G}$ do
3      foreach $L \to R \in \mathcal{L}$ such that $L \cup R \cup G \subseteq P$ do
4          $S \leftarrow L \cup (K \setminus R)$
5          $flag \leftarrow true$
6          foreach $H \in \mathcal{G}$ do
7              if $H \subseteq S$ then
8                  $flag \leftarrow false$
9              end
10         end
11         if $flag$ then
12             $\mathcal{G} \leftarrow \mathcal{G} \cup \{MinGen(S, \mathcal{L})\}$
13         end
14     end
15 end
```

Algorithm 5: All minimal transversals

```
Input  : Hypergraph $\mathcal{H}$
Output: $Tr(\mathcal{H})$
1  $T = \emptyset$
2  foreach $E \in \mathcal{H}$ do
3      $T = min(T \vee \{\{v\} \mid v \in E\})$
4  end
5  Return $T$
```

NextClosure on formal contexts $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ randomly generated with 50 objects, 12 attributes and a probability d (called *density*) to have $(o, a) \in \mathcal{R}$.

Two cases were considered:

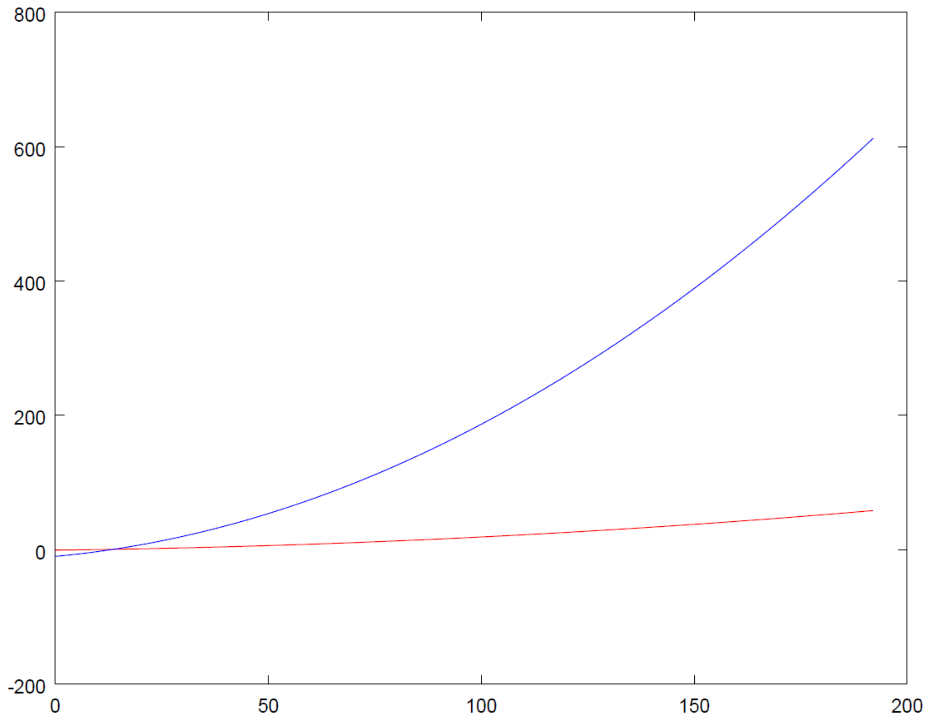
– Single computations of lower covers
– Depth-first searches in lattices involving multiple computations of lower covers

For the first case, we simply computed the lower covers of $\mathcal{A}$. For the second case, we removed some implications as to introduce, in the lattice, sets that are not intents of the formal contexts and we performed the depth-first searches starting from $\mathcal{A}$ and going deeper everytime we encountered a non-intent.

5.1 Single Computations

We generated 1000 contexts with varying numbers of implications and computed the lower covers of $\mathcal{A}$ using the two methods we presented. Figure 1 presents the quadratic interpolation of the runtimes for each method relative to the number of implications in the basis.

Fig. 1. Runtimes for both algorithms relative to the number of implications on a single computation (Red = Meet-irreducibles, Blue = Minimal generators)



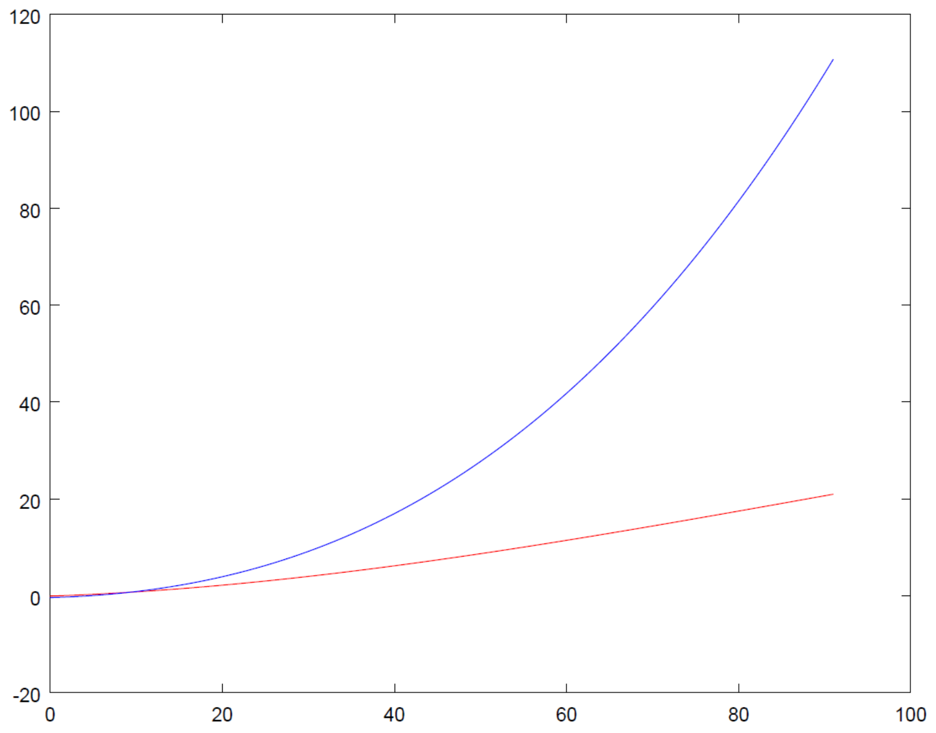
In our experiments, the algorithm using meet-irreducible elements outperformed the one based on minimal generators 79% of the time. Computing the meet-irreducible elements represents 98% of its runtime. The second method devotes 75% of its time to computing minimal generators and 25% on minimal transversals.

5.2 Depth-first Searches

As with single computations, we randomly generated 1000 contexts with varying numbers of implications and performed depth-first searches starting from $\mathcal{A}$.

Runtimes Relative to the Size of the Basis Figure 2 presents the interpolation of the runtimes for each methods relative to the number of implications in the basis.

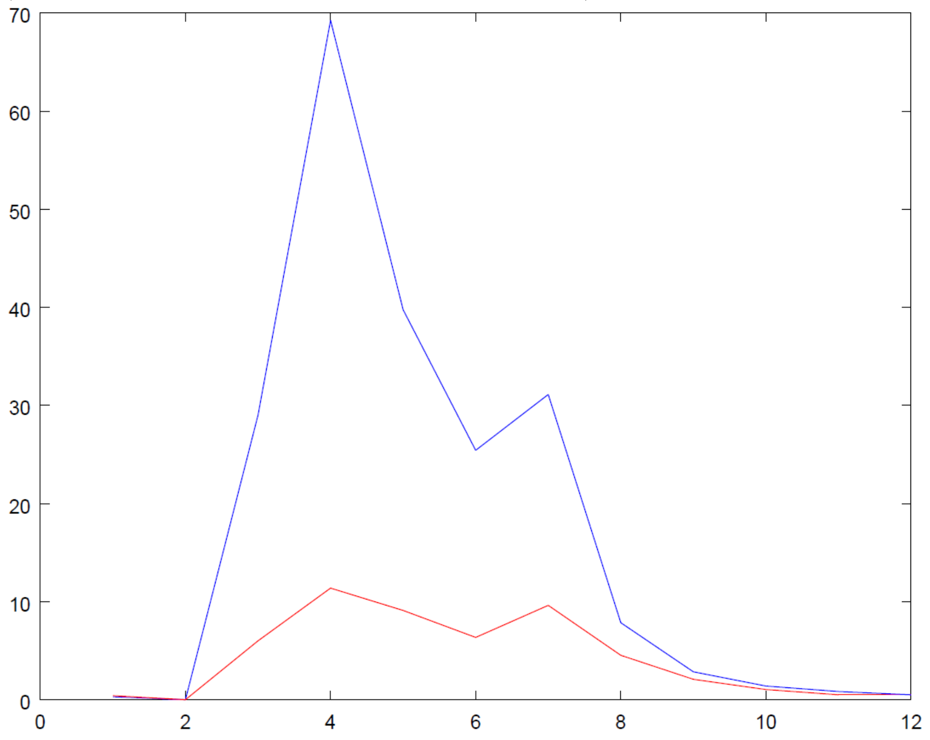
Fig. 2. Runtimes for both algorithms relative to the number of implications for depth-first searches (Red = Meet-irreducibles, Blue = Minimal generators)



Once again, using meet-irreducibles is the most efficient method (it outperformed the minimal generators 72% of the time) and this trend accentuates with the size of the implicational basis. Computing the meet-irreducible elements represents 86% of the total runtime. The second methods devotes 69% of its runtime to the computation of minimal generators and 31% on minimal transversals.

Runtimes Relative to the Depth Figure 3 presents the interpolation of the runtimes for each methods relative to the depth of the search.

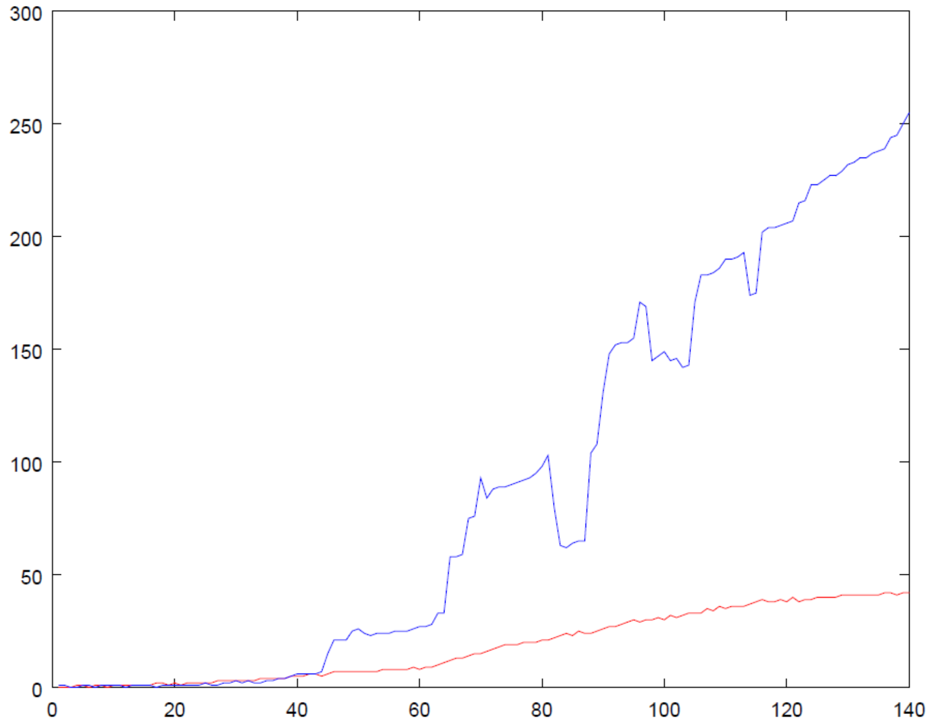
Fig. 3. Runtimes for both algorithms relative to the maximum depth of the search (Red = Meet-irreducibles, Blue = Minimal generators)



We can notice that the algorithm using meet-irreducible elements is much less affected by the depth of the search, most likely due to the fact that most of the computation is done prior to the actual search. Interestingly, the algorithm using minimal generators is least efficient for relatively shallow searches. We believe this is due to the fact that deep searches imply that the lattice is mostly composed of sets that have the property we are looking for. As we used closedness relative to a formal context, this means that those lattices correspond to nearly empty implicational bases.

Runtimes with Increasing Bases We used both methods to perform depth-first searches in a sequence of lattices corresponding to implicational bases $\emptyset \subseteq \mathcal{I}_1 \subseteq ... \subseteq \mathcal{I}_n$. Results are presented in Figure 4.

Fig. 4. Runtimes for both algorithms relative to a growing implicational basis (x-axis = number of implications)(Red = meet-irreducibles, Blue = Minimal generators)



Once again, the runtime of the algorithm using minimal generators skyrockets when the basis grows. The meet-irreducibles method presents much less variance. It should be noted that the meet-irreducible elements are computed from scratch for each basis when Algorithm 1 could be used to speed up the process by using the meet-irreducible elements of the previous lattice to compute the new ones.

6 Discussion

Experimental results indicate that the most efficient way to compute lower covers of implication-closed sets is to first compute the meet-irreducible elements of the lattice and then intersect them. It outperforms the method based on computing minimal transversals of minimal generators even when minimal generators have to be computed only once. During deeper searches, minimal generators have to be computed at each step and the difference is thus more pronounced.

While the algorithm we used for computing minimal transversals is not the most efficient, the fact that this computation represents only a small portion of the second method indicates that, even with the best algorithm, using meet-irreducible elements would be best. Computing these meet-irreducible elements is time-consuming but, once we have them, they are easy to intersect. We believe there is still more room for improvement on the problem of computing meet-irreducible elements from implicational bases.

Acknowledgments The author was supported by the European Union and the French Région Auvergne - Rhône-Alpes.

References

1. Mikhail A. Babin and Sergei O. Kuznetsov. Dualization in lattices given by ordered sets of irreducibles. *Theoretical Computer Science, 2016. CoRR*, abs/1504.01145, 2015.
2. Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45. Elsevier, 1984.
3. Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM J. Comput.*, 32(2):514–537, 2003.
4. Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational Aspects of Monotone Dualization: A Brief Survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.
5. Michael L. Fredman and Leonid Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *J. Algorithms*, 21(3):618–628, 1996.
6. Bernhard Ganter and Klaus Reuter. Finding all Closed Cets: A General Approach. *Order*, 8(3):283–290, 1991.
7. Miki Hermann and Baris Sertkaya. On the Complexity of Computing Generators of Closed Sets. In Raoul Medina and Sergei A. Obiedkov, editors, *ICFCA*, volume 4933 of *Lecture Notes in Computer Science*, pages 158–168. Springer, 2008.
8. Petr Krajca, Jan Outrata, and Vilém Vychodil. Advances in Algorithms Based on CbO. In Marzena Kryszkiewicz and Sergei A. Obiedkov, editors, *CLA*, volume 672 of *CEUR Workshop Proceedings*, pages 325–337. CEUR-WS.org, 2010.
9. Marcel Wild. Computations with Finite Closure Systems and Implications. In Ding-Zhu Du and Ming Li 0001, editors, *COCOON*, volume 959 of *Lecture Notes in Computer Science*, pages 111–120. Springer, 1995.