Horn Minimization

Vilmin Simon

Higher School of Economics - Faculty of Computer Science

1 Abstract

The topic is implication theories, or Horn, minimization, used in database applications for instance. More precisely the aim is to provide a review of existing algorithms for performing minimization and implement them to see how do they behave under practical test cases.

We study several algorithms and review them within the context of closure systems. On top of providing explanations on their operation and complexity analysis, we implemented those algorithms using C++. With randomly generated systems and real data, we supply informations on which algorithm or steps are likely to be used in practice. Those results being valid within the context of our tests, suggest further research and experiments to lead in future work.

2 Introduction

The question of minimization has been discussed and developed through various frameworks, and several computer scientists communities. Notice that in order not to make this synthesis too long, we will stay within the context of minimization and will not trace the field of implication theories in general. For a survey of this domain anyway, the reader should refer to [30]. Also, note that minimality in general terms is not unique. Indeed, one can define several types of minimality among implication systems. For instance, not only we can define minimality with respect to the number of implications within a system (which is our interest) but also with respect to the number of attributes in each implications. The former one is called canonical in relational database field, and hyperarc minimum within the graph context. Especially in the graph-theoretic and boolean logic settings, one can derive more types of minimality. For general introduction to boolean logic notations, we invite the reader to see [15]. In terms of propositional logic, implications are represented through Horn formulae. Interestingly, the minimization problem we are going to consider is the only one being polynomial time solvable. Other problems are proved to be NP-Complete or NP-Hard. For more discussion on other minimality definitions and their computational complexity, the reader should refer to [13], [7], [20], [6], [30], [11], [23]. In subsequent explanations, we will refer to minimization with respect to the number of implications.

To the best of our knowledge, the two first fields in which algorithms and properties of minimality arose are Formal Concept Analysis (FCA) (see [21],

[19] for an introduction) and Database Theory (DB) (see [25]). Both sides were developed independently in the early 80's. For the first domain, characterization of minimality goes to Duquenne and Guigues [22], in which they describe the so-called canonical basis (also called Duquenne-Guigues basis after its authors, abbreviated DG all along this report) relying on the notion of pseudo-closed sets. For the database part, study of implications is made by Maier through functional dependencies ([25], [24]). The polynomial time algorithm he gives for minimization heavily relies on a fast subroutine discovered by Beeri and Bernstein in [9], 1979.

From then on, knowledge increased over years and spread out over domains. Another algorithm based on a minimality theorem is given by Shock in 1986 ([26]). Unfortunately, as we shall see and as already discussed by Wild in [29] the algorithm may not be correct in general, even though the underlying theorem is. During the same period, Ausiello et al. brought the problem to graph-theoretic ground, and provided new structure known as FD-Graph and algorithm to represent and work on implication systems in [7], [5], [6]. This approach has been seen in graph theory as an extension of the transitive closure in graphs ([1]), but no consideration equivalent to minimization task seems to have been taken beforehand, as far as we know. Still in the 1980 decade, Ganter expressed the canonical basis formalized by Duquenne and Guigues in his paper related to algorithms in FCA, [19] through closure systems, pseudo-closed and quasi-closed sets. Also, the works of Maier and Duquenne-Guigues have been used in the lattice-theoretic context by Day in [17] to derive an algorithm based on congruence relations. For in-depth knowledge of implication system within lattice terminology, we can see [16] as an introduction and [10] for a survey. Next, Wild ([27], [28], [29]) linked within this set-theoretic framework both the relational databases, formal concept analysis and lattice-theoretic approach. In relating those fields, he describes an algorithm for minimizing a basis, similar to algorithms of Day and, somehow, Shock (resp. [17], [26]). This framework is the one we will use for our study, and can be found in more recent work by Ganter & Obiedkov in [20]. Later, Duquenne proposed some variations of Day's work with another algorithm in [18]. More recently, Boròs et al. by working in a boolean logic framework, exhibited a theorem on the size of canonical basis [12], [13]. They also gave a general theoretic approach that algorithm should do one way or another on reduction purpose. Out of these papers, Berczi & al. derived a new minimization procedure based on hypergraphs in [14]. Furthermore, an algorithm for computing the canonical basis starting from any system is given in [20]. This last algorithm was our starting point for this study.

Even though the work we are going to cite is not designed to answer this question of minimization, it must also be exposed as the algorithm is intimately related to DG basis and can be used for base reduction. The paper of Angluin et al. in query learning, see [2], provides an algorithm for learning a Horn representation of an unknown initial formula. It has been shown later by Ariàs and Alcazar ([3]) that the output of Angluin algorithm was always the Duquennes-Guigues basis.

In this paper our aim is to review various algorithms dedicated to the minimization task. After description of our notations and recall of the main definitions, we present different algorithms. Eventually, we exhibit some experiments on those algorithms.

3 Notations

For our study let us define notions. Σ will be an attribute set. An implication is an ordered pair $(A, B), A, B \subseteq \Sigma$ denoted $A \longrightarrow B$. A is the premise, B the conclusion of the implication. A set of implications $\mathcal{L}$ over Σ is an implication system/theory. A subset M of Σ is a model of $A \longrightarrow B$, denoted $M \models A \longrightarrow B$ if $B \subseteq M \vee A \nsubseteq M$. M is a model of an implication system $\mathcal{L}$ if it is a model for all implications of $\mathcal{L}$. In fact, models of $\mathcal{L}$ define a closure system $\Sigma^{\mathcal{L}}$.

Definition 1 (Closure operator). *Let Σ be a set and $\phi : 2^{\Sigma} \longrightarrow 2^{\Sigma}$ an application on the power set of Σ. ϕ is a closure operator if $\forall X, Y \subseteq \Sigma$:*

(i) $X \subseteq \phi(X)$ *(extensive),*
(ii) $X \subseteq Y \longrightarrow \phi(X) \subseteq \phi(Y)$ *(monotone),*
(iii) $\phi(X) = \phi(\phi(X))$ *(idempotent).*

$X \subseteq \Sigma$ is called closed if $X = \phi(X)$.

Definition 2 (Closure system). *Let Σ be a set, and $\Sigma^{\phi} \subseteq 2^{\Sigma}$. Σ^{ϕ} is called a closure system if:*

(i) $\Sigma \in \Sigma^{\phi}$,
(ii) if $\mathcal{S} \subseteq \Sigma^{\phi}$, *then* $\bigcap \mathcal{S} \in \Sigma^{\phi}$ *(closed under intersection).*

With this closure system comes a closure operator $\mathcal{L}(\cdot)$ associating to any $X \subseteq \Sigma$, the smallest model of $\Sigma^{\mathcal{L}}$ containing X:

$$\mathcal{L}(X) = \bigcap \{A \in \Sigma^{\mathcal{L}},\ X \subseteq A\}$$

$A \longrightarrow B$ semantically follows from $\mathcal{L}$, $\mathcal{L} \models A \longrightarrow B$ if all models of $\mathcal{L}$ are models of $A \longrightarrow B$. More generally, if we have two systems $\mathcal{L}_1, \mathcal{L}_2$, $\mathcal{L}_1 \models \mathcal{L}_2$ if all implications of $\mathcal{L}_2$ follow from $\mathcal{L}_1$. Checking entailment may seem to be expensive in terms of set operations. Fortunately, a proposition based on closure operators ease the way.

Proposition 1. *$\mathcal{L} \models A \longrightarrow B$ if and only if $B \subseteq \mathcal{L}(A)$.*

Two systems $\mathcal{L}_1$, $\mathcal{L}_2$ are equivalent if they define the same closure system. More precisely, they are equivalent if $\mathcal{L}_1 \models \mathcal{L}_2$ and $\mathcal{L}_2 \models \mathcal{L}_1$. Given some $\mathcal{L}$ over Σ one can find *"useless"* implications, or redundant implications. An implication $A \longrightarrow B \in \mathcal{L}$ is redundant if $\mathcal{L} - \{A \longrightarrow B\} \models A \longrightarrow B$. For the remainder of the paper, we may denote by $\mathcal{L}^{-}$ the system $\mathcal{L} - \{A \longrightarrow B\}$. It should be clear when used.

Definition 3 (Minimum base). *$\mathcal{L}$ is a minimum base if there is no equivalent system $\mathcal{L}'$ with fewer implications than $\mathcal{L}$.*

A well known minimum base is the canonical or Duquennes-Guigues ([22]). It relies on pseudo-closed sets.

Definition 4 (Pseudo-closed set). *Given $\mathcal{L}$ over Σ, we say that $P \subseteq \Sigma$ is pseudo-closed if:*

(i) $P \neq \mathcal{L}(P)$,
(ii) $Q \subset P$ *and* Q *pseudo-closed implies* $\mathcal{L}(Q) \subseteq P$.

Definition 5 (Quasi-closed set). *a set $Q \subseteq \Sigma$ is quasi-closed with respect to $\mathcal{L}$ if $\forall A \subseteq Q, \mathcal{L}(A) \subseteq Q$ or $\mathcal{L}(A) = \mathcal{L}(Q)$*

Definition 6 (Duquenne-Guigues basis). *The base $\mathcal{L}$ defined by*

$$\mathcal{L} = \{P \longrightarrow \mathcal{L}(P) \mid P \text{ is pseudo-closed in } \mathcal{L}\}$$

is called the Duquenne-Guigues or canonical basis. It is minimum.

At first sight, quasi-closed set may not be useful. However, one can define pseudo-closed sets out of closed sets. A set P is pseudo-closed if and only if it is quasi-closed and minimal (inclusion wise) among quasi-closed sets having the same closure. This property will be useful for some algorithms we will explain. So far we discussed several notions: implications, pseudo-closed set, quasi-closed set, canonical basis and so forth. Most of them rely heavily on computing the closure of sets with respect to $\mathcal{L}$. Hence, to have practical efficiency, we must be able to compute closures as fast as possible. Fortunately, several algorithms can be found. Among them, there is a naïve procedure based on the operation $\circ$:

$$X^\circ = X \cup \bigcup \{B \mid A \longrightarrow B \in \mathcal{L},\ A \subseteq X\}$$

Repetition of this operation up to saturation leads to Closure. Furthermore the algorithm by Beeri and Bernstein in [9] called LinClosure addresses this question. LinClosure as previously mentioned has been widely used, notably in [25], [24], [20], [26], [17]. Before describing those procedures, let us introduce our complexity notations:

- $|\Sigma|$ will denote the size of the attribute set Σ,
- $|\mathcal{B}|$ will be the number of implications in $\mathcal{L}$ ($\mathcal{B}$ stands for body),
- $|\mathcal{L}|$ is the number of symbols used to represent $\mathcal{L}$.

Recall that O is the asymptotically worst case complexity (in time or space). For instance, in the worst case, $|\mathcal{L}| = |\mathcal{B}| \times |\Sigma|$, thus $|\mathcal{L}| = O(|\mathcal{B}| \times |\Sigma|)$. Closure and LinClosure are algorithms 1, 2 (resp.).

As we already mentioned, the algorithm Closure relies on the $\circ$ operation. The principle is to re-roll over the set of implications $\mathcal{L}$ to see whether there

Algorithm 1: CLOSURE

Input: A base $\mathcal{L}$, $X \subseteq \Sigma$
Output: The closure $\mathcal{L}(X)$ of X under $\mathcal{L}$

```
closed := ⊥ ;
L(X) := X ;
while ¬closed do
    closed := ⊤ ;
    foreach A ⟶ B ∈ L do
        if A ⊆ L(X) then
            L(X) := L(X) ∪ B ;
            L := L −{A ⟶ B} ;
            closed := ⊥ ;
return L(X);
```

exists an implication $A \longrightarrow B$ in $\mathcal{L}$ such that $\mathcal{L}(X) \not\models A \longrightarrow B$ up to stability. Asymptotically, we will need $O(|\,\mathcal{B}\,|^2 \times |\,\Sigma\,|)$ if we remove only one implication per loop. the $|\,\Sigma\,|$ cost comes from the set union. $\bot$ stands for *false* and $\top$ for *true*.

LINCLOSURE has $O(|\,\mathcal{L}\,|)$ time complexity. The main idea is to use counters. Starting from X, if we reach for a given $A \longrightarrow B$ as many elements as $|A|$, then $A \subseteq \mathcal{L}(X)$ and we must also add B. Because the closure in itself is not the main point of our topic, we will not study LINCLOSURE in depth. Furthermore, there exists other algorithm for computing closure given by Wild in [29]. It is derived from LINCLOSURE, but we did not consider it because it brings no improvement in complexity. For more complete theoretical and practical comparisons of closure algorithms, we redirect the reader to [8]. In this paper, LINCLOSURE is shown maybe not to be the most efficient algorithm in practice when used in other algorithms, especially when compared with CLOSURE. Anyway, because of its theoretical complexity and use in all algorithms we will review, we will still consider LINCLOSURE.

4 Algorithms for minimization

We are going to focus on 5 algorithms. To begin with, let us talk about an algorithm issued by A. Day in [17] and by Wild somehow in [27]. It can be found as we will write it in [20]. The principle is to perform right-closure first so has to have right-closed implications. Next, it computes quasi-closure to find pseudo-closed sets and remove redundant implications. See algorithm 3.

Given the linear complexity $O(|\,\mathcal{L}\,|)$ of LINCLOSURE being the theoretically fastest closure algorithm, both loops of MINCOVER require at most $O(|\,\mathcal{B}\,||\,\mathcal{L}\,|)$ operations. Hence $O(|\,\mathcal{B}\,||\,\mathcal{L}\,|)$ is the complexity of the all algorithm. In the paper of Day, this algorithm is discussed in terms of congruences within complete join-semilattices, being much more algebraical.

Algorithm 2: LinClosure

Input: A base $\mathcal{L}$, $X \subseteq \Sigma$
Output: The closure $\mathcal{L}(X)$ of X under $\mathcal{L}$

foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  $count[A \longrightarrow B] := |A|$;
  if $|A| = 0$ **then**
    $X := X \cup B$;
  foreach $a \in A$ **do**
    $list[a] = list[a] \cup \{A \longrightarrow B\}$;

$update := X$;

while $update \neq \emptyset$ **do**
  choose $m \in update$;
  $update := update - \{m\}$;
  foreach $A \longrightarrow B \in list[m]$ **do**
    $count[A \longrightarrow B] := count[A \longrightarrow B] - 1$;
    if $count[A \longrightarrow B] = 0$ **then**
      $add := B - X$;
      $X := X \cup add$;
      $update := update \cup add$;

return X ;

Algorithm 3: MinCover

Input: $\mathcal{L}$: an implication system
Output: the canonical base of $\mathcal{L}$

foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  $\mathcal{L} := \mathcal{L} - \{A \longrightarrow B\}$;
  $B := \mathcal{L}(A \cup B)$;
  $\mathcal{L} := \mathcal{L} \cup \{A \longrightarrow B\}$;

foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  $\mathcal{L} := \mathcal{L} - \{A \longrightarrow B\}$;
  $A := \mathcal{L}(A)$;
  if $A \neq B$ **then**
    $\mathcal{L} := \mathcal{L} \cup \{A \longrightarrow B\}$;

The second algorithm we have has been discussed by Duquenne in [18] as variations of Day algorithms. In particular, one can find in DuquenneMinimization (see algorithm 4) an alternative to MinCover.

Algorithm 4: DuquenneMinimization

Input: $\mathcal{L}$ a theory to minimize
Output: $\mathcal{L}_c$ the DQ-basis of $\mathcal{L}$

foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  $\mathcal{L} = \mathcal{L} - \{A \longrightarrow B\}$;
  $A := \mathcal{L}(A)$;
  if $B \not\subseteq A$ **then**
    $B = B \cup A$;
    $\mathcal{L} := \mathcal{L} \cup \{A \longrightarrow B\}$;

LecticOrder($\mathcal{L}$) ;
$\mathcal{L}_c := \emptyset$;
foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  foreach $\alpha \longrightarrow \beta \in \mathcal{L}_c$ **do**
    if $\alpha \subset A \wedge \beta \not\subseteq A$ **then**
      $\mathcal{L} = \mathcal{L} - \{A \longrightarrow B\}$;
      goto next $A \longrightarrow B \in \mathcal{L}$;
  $B = \mathcal{L}(B)$;
  $\mathcal{L}_c := \mathcal{L}_c \cup \{A \longrightarrow B\}$;

return $\mathcal{L}_c$;

In this algorithm, we first perform left-quasi-closure as much as redundancy elimination. Doing those steps not only allows for having quasi-closed, and in particular all possible pseudo-closed sets, as premises of implications but also to get rid of several useless ones. Next, ordering of implications in a $\subseteq$-compatible way on premises ease the last step. We use lectic ordering $<_\Sigma$ being total. It assumes that elements of Σ are linearly ordered (e.g: $\Sigma = \{a, b, c, d\}$ and $a < b < c < d$) and performs somehow binary enumeration of characteristic vectors over Σ (in our example we would have in order: $\emptyset$, d, c, cd, b, bd, bc, bcd, etc). The last step of DuquenneMinimization consists in iteratively build the canonical base $\mathcal{L}_c$ as an output. Because we have quasi-closed premises under lectic order, checking for pseudo-closedness for any premise of the input system only requires a run over the base we are building. If a premise is to be considered pseudo-closed, its corresponding implication is right-closed and added to the resulting system ($\mathcal{L}_c$). Regarding the complexity, still thinking of LinClosure, the first loop, quite similar to the second one of MinClosure requires $O(|\,\mathcal{B}\,||\,\mathcal{L}\,|)$ operations. Because lectic order is total, one can use fast sorting procedure to perform the second step: $O(|\,\mathcal{L}\,|\log(|\,\mathcal{B}\,|))$. Finally, the building loop may require no more

than $O(|\mathcal{B}||\mathcal{L}|)$ operations since closure computations occur out of the nested loop. Consequently, the whole algorithm runs in $O(|\mathcal{B}||\mathcal{L}|)$ as MinCover and ends up on the Duquenne-Guigues basis too.

Next, we are interested in an algorithm quite different since coming out of Database theory. It has been proposed by Maier in [24], [25]. Wild discussed it and its connection with closure spaces framework in [27]. In our case, we keep on focusing on implications notations, see MaierMinimization. Here, apart from prior redundancy elimination as previously studied, we are to split implications into equivalence classes according to the closure of their premises. Then, using *direct determination*, equivalence classes will be reduced. In details, an implication $A \longrightarrow B$ will be removed when one can find an implication $C \longrightarrow D$ such that $A \longrightarrow B$, $C \longrightarrow D \in E_{\mathcal{L}}(A)$ and $\mathcal{L} - E_{\mathcal{L}}(A) \models A \longrightarrow C$, where $E_{\mathcal{L}}(A) = \{C \longrightarrow D \in \mathcal{L} \mid \mathcal{L}(C) = \mathcal{L}(A)\}$. In this case, A directly determines C and $A \longrightarrow B$ can be removed if we replace $C \longrightarrow D$ by $C \longrightarrow B \cup D$ to preserve the closure system defined by $\mathcal{L}$. The algorithm ends up on a minimum cover different from the canonical basis. Indeed, observe that we do not alter premises of implications, hence they may not be pseudo-closed.

Algorithm 5: MaierMinimization

Input: $\mathcal{L}$: a theory to minimize
Output: $\mathcal{L}$ minimized

foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  if $\mathcal{L} - \{A \longrightarrow B\} \models A \longrightarrow B$ **then**
    remove $A \longrightarrow B$ from $\mathcal{L}$;

$E_{\mathcal{L}} :=$ EquivClasses($\mathcal{L}$) ;

foreach $E_{\mathcal{L}}(X) \in E_{\mathcal{L}}$ **do**
  foreach $A \longrightarrow B \in E_{\mathcal{L}}(X)$ **do**
    if $\exists C \longrightarrow D \in E_{\mathcal{L}}(X)$ *s.t* $A \xrightarrow{\text{d}} C$ **then**
      remove $A \longrightarrow B$ from $\mathcal{L}$;
      replace $C \longrightarrow D$ by $C \longrightarrow D \cup B$;

Let us focus on the complexity of this algorithm. Redundancy elimination can be done in $O(|\mathcal{B}||\mathcal{L}|)$ operations. To determine equivalence classes, the idea provided by Maier is to alter LinClosure. For each implication $A \longrightarrow B$, the closure operator permits to get all other premises reachable from A. When this $O(|\mathcal{B}||\mathcal{L}|)$ operation is done, we have a $|\mathcal{B}| \times |\mathcal{B}|$ matrix. With this structure, determining equivalence classes requires no more than a run over it hence $O(|\mathcal{B}|^2)$ operations. Finally, finding direct determination is again an alteration of LinClosure. For each implication $A \longrightarrow B$, it is sufficient to stop closure computation when we reach a premise of $E_{\mathcal{L}}(A)$. Not omitting subsequent set union,

the whole loop needs $O(|\mathcal{B}||\mathcal{L}|)$ operations to terminate. The whole complexity of the algorithm is then $O(|\mathcal{B}||\mathcal{L}|)$.

More recently, an algorithm has been issued by Berczi et al. in [14].Contrary to the previous algorithms, in this case we are likely to build the canonical base instead of minimizing the input system. We Refer to algorithm 6. Originally it is logic/hypergraph based, but we give it in terms of implications and closures. Starting from an empty system $\mathcal{L}_c$, and up to equivalence with the input $\mathcal{L}$, we iteratively take the next minimal pseudo-closed set P of $\mathcal{L}$ (inclusion-wise) and add the implication $P \longrightarrow \mathcal{L}(P)$ to $\mathcal{L}_c$. This way, the algorithm terminates on the Duquenne-Guigues base.

Algorithm 6: BERCZIMINIMIZATION

Input: $\mathcal{L}$: an implication theory
Output: $\mathcal{L}_c$: the DG basis of $\mathcal{L}$

$\mathcal{L}_c := \emptyset$;
while $\exists B \in \mathcal{B}(\mathcal{L})$ *s.t* $\mathcal{L}_c(B) \neq \mathcal{L}(B)$ **do**
  $P := \min\{\mathcal{L}_c(B),\ B \in \mathcal{B}(\mathcal{L})$ and $\mathcal{L}_c(B) \neq \mathcal{L}(B)\}$;
  $\mathcal{L}_c := \mathcal{L}_c \cup \{P \longrightarrow \mathcal{L}(P)\}$;
return $\mathcal{L}_c$;

Here however, we must face a higher complexity than previously exposed algorithm. The outter loop runs up to equivalence, but since we add an implication to $\mathcal{L}_c$ per iteration, we must end after at most $|\mathcal{B}|$ steps. Finding the next minimum pseudo-closed sets require closure computations under both $\mathcal{L}$ and $\mathcal{L}_c$ for all premises of $\mathcal{L}$, hence $O(|\mathcal{B}||\mathcal{L}|)$ operations. Therefore, the algorithm has $O(|\mathcal{B}|^2|\mathcal{L})$ time complexity.

Finally, let us focus on an algorithm derived from Angluin query-learning based approach (see [2], [3]). For a quick recap, the idea is we formulate queries to an oracle knowing the basis we are trying to learn. The oracle is assumed to provide an answer to our query in constant time. Depending on the query, it might also provide information on the object we are looking for. For Angluin algorithm, we need 2 types of queries. Say we want to learn a basis $\mathcal{L}$ over Σ:

1. membership query: is $M \subseteq \Sigma$ a model of $\mathcal{L}$? The oracle may answer *"yes"*, or *"no"*.
2. equivalence query: is a basis $\mathcal{L}'$ equivalent to $\mathcal{L}$? Again the answers are *"yes"*, or *"no"*. In the second case, the oracle provides a counterexample either positive or negative:
   (i) positive: a model M of $\mathcal{L}$ which is not a model of $\mathcal{L}'$,
   (ii) negative: a non-model M of $\mathcal{L}$ being a model of $\mathcal{L}'$.

To clarify, the terms negative/positive are related to the base $\mathcal{L}$ we want to learn. ANGLUINALGORITHM (7) is the algorithm presented by Angluin, Frazer

and Pitts in [2] as HORN1. Initially, it is based on learning logical representation of implication theories: Horn clauses. This learning algorithm has been shown first to terminate on a minimum representation of the basis we want to learn ([2]) and more than that, to end up on the canonical one by Arias et al. [3]. It uses two operations allowing to reduce implications:

- *refine*$(A \longrightarrow B, M)$: produces $M \longrightarrow \Sigma$ if $B = \Sigma$, $M \longrightarrow B \cup A - M$ otherwise,
- *reduce*$(A \longrightarrow B, M)$: produces $A \longrightarrow M - A$ if $B = \Sigma$, $A \longrightarrow B \cap M$ otherwise.

The main idea is to ask the oracle whether the theory we are building ($\mathcal{L}_c$) is equivalent to $\mathcal{L}$ until it answers *"yes"*. If it says *"no"* then it provides an example on which $\mathcal{L}_c$ and $\mathcal{L}$ differ. If the example is a model of $\mathcal{L}$, then we track implications in $\mathcal{L}_c$ falsified by this example and correct them. If the example however is not a model of $\mathcal{L}$ we look for the first possible smaller example out of the one we got. The main idea is to say that if we correct a smaller example, we are likely to correct a larger one too. If we do not find any smaller example to correct, we add an implication in $\mathcal{L}_c$ addressing the problem. In practice, we may not be given the power of an oracle. Therefore, one can derive from ANGLUINALGORITHM the algorithm AFPMINIMIZATION (8).

Algorithm 7: ANGLUINALGORITHM

Input: $\mathcal{L}$ a theory to learn and an *Oracle* with *membership, equivalence* queries
Output: $\mathcal{L}_c$ the canonical representation of $\mathcal{L}$

$\mathcal{L}_c = \emptyset$;
while *not equivalence*$(\mathcal{L}_c)$ **do**
  M is the counterexample ;
  if M *is positive* **then**
    foreach $A \longrightarrow B \in \mathcal{L}_c$ *such that* $M \not\models A \longrightarrow B$ **do**
      replace $A \longrightarrow B$ by $reduce(A \longrightarrow B, M)$;
  else
    foreach $A \longrightarrow B \in \mathcal{L}_c$ *such that* $A \cap M \subset A$ **do**
      membership$(M \cap A)$;
    if *Oracle replied "no" for at least one* $A \longrightarrow B$ **then**
      Take the first such $A \longrightarrow B$ in $\mathcal{L}_c$;
      replace $A \longrightarrow B$ by $refine(A \longrightarrow B, A \cap M)$;
    else
      add $M \longrightarrow \Sigma$ to $\mathcal{L}_c$;
return $\mathcal{L}_c$;

In this version, we use a stack to keep track of possible generators of negative counter-example. Whenever we find one, we apply the same operations as in AFP. Note that we do not use positive counter-example since we only consider right-closed implications. Unfortunately we are not yet able to prove the

Algorithm 8: AFPMinimization

Input: some theory $\mathcal{L}$ over Σ
Output: $\mathcal{L}_c$ the Duquenne-Guigues basis of $\mathcal{L}$

```
L_c := ∅ ;
Stack S ;
forall A ⟶ B ∈ L do
    S := [A] ;
    repeat
        X := L_c(pop(S)) ;
        if X ≠ L(X) then
            found := ⊥ ;
            forall α ⟶ β ∈ L_c do
                C := α ∩ X ;
                if C ≠ α then
                    D := L(C) ;
                    if C ≠ D then
                        found := ⊤ ;
                        change α ⟶ β by C ⟶ D in L_c;
                        push(X ∪ D, S) ;
                        if β ≠ D then
                            push(α, S);
                        exit for
            if found = ⊥ then
                L_c := L_c ∪ {X ⟶ L(X)} ;
    until S = ∅;
return L_c ;
```

algorithm nor give a precise complexity. Assuming the stack does not store more than $|\mathcal{B}|$ examples, one may find at most $O(|\mathcal{B}|^3|\mathcal{L}|)$ time complexity.

Note that BerczıMinimization can be interpreted in terms of query learning too. In fact, the algorithm is ruled by an equivalence query. At each step we consider the next minimal negative counter-example being pseudo-closed. Because we base the algorithm on premises of $\mathcal{L}$, we conclude that premises of the input system $\mathcal{L}$ can generate a sufficient set of negative counter examples when taken as in BerczıMinimization. From this point of view, it gets closer to Angluin algorithm.

5 Experiments

Those algorithms have been implemented in C++, using MinGW-64 on a 2.4GHz CPU. We used the code given in https://github.com/yazevnul/fcai developed for the benchmark of closure operators (see [8]). It relies on a formal concept analysis background. Let us remind some definitions of FCA. Formal Concept Analysis (see [21]) is a technique relying on array-like data and lattices to describe hierarchies in data. It can be used in data mining, text mining or chemistry for instance. Usually, we are given a context $\mathbb{K} = (G, M, I)$ where G is a set of objects, M a set of attributes and $I \subseteq G \times M$ a relation between them. We can define an operation $' : 2^G \longrightarrow 2^M$, associating to a set of objects $A \subseteq G$ the set $B \subseteq M$ of attributes shared by all elements of A. Conversely, we can set $' : 2^M \longrightarrow 2^G$ yelling the set A of objects sharing all attributes of some set B. Formally:

$$A' = \{m \in M \mid \forall g \in A,\ gIm\} \quad \forall A \subseteq G$$
$$B' = \{g \in G \mid \forall m \in B,\ gIm\} \quad \forall B \subseteq M$$

When combined together those operators define a closure operator $'' : 2^M \longrightarrow 2^M$. Pairs (A_G, A_M) of closed subsets from $G \times M$ with $A'_G = A_M$ and $A'_M = A_G$ are called concepts. A_G is the concept extent while A_M is called concept intent. As one could expect, this operator is related with attribute implications. Indeed, given a context (G, M, I) we can draw implications between subsets $A \longrightarrow B$ of attributes. Intuitively, an implication $A \longrightarrow B$ will be valid if every time we have all attributes from A, we also have attributes from B: $A' \subseteq B'$ or equivalently, $B \subseteq A''$. A set $\mathcal{L}$ of implications over M will be complete and sound if the operator $''$ from the context coincides with $\mathcal{L}(\cdot)$ for all $A \subseteq M$. Because real datasets used contain multi-valued attributes, we may need FCA-scaling. Eventually, we shall discuss various possible systems one can build out of a context. We will use 5 of them: the canonical basis, the minimum basis resulting from Maier's algorithm (twice), the basis of minimal generators and the proper basis. We already discussed the two first ones. We derive them from the non minimum:

(i) minimal generators: a set X is a minimal generator of its closure $\mathcal{L}(X)$ if it is minimal in $[X]_{\mathcal{L}}$. The basis will contain right-closed implications $X \longrightarrow \mathcal{L}(X)$ where X is a minimal generator,

(ii) proper implications: implications $X \longrightarrow X^\bullet$ where $\bullet$ is a saturation operator:

$$X^\bullet = \mathcal{L}(X) - X \cup \bigcup \{\mathcal{L}(B) \mid B \subset X\}$$

in words, $X^\bullet$ is the set of attributes in the closure of X to which no proper subset of X can lead. X is the *"minimal"* set required to get those attributes in $\mathcal{L}(X)$. We redirect the reader to [20] for more details. If $X^\bullet = \mathcal{L}(X) - X$, $X^\bullet$ is pseudo-closed. It is also used in [18] to prove DuquenneMinimization.

We use the theory produced by MaierMinimization on both proper and minimal generator basis since the results may differ. algorithms to compute those basis from a context are included in the code we got from `https://github.com/yazevnul/fcai`. We give pseudo-code of the algorithms here, but we will not review them in details. MininalGeneratorBasis (algorithm 9) generates minimal generators. recall that $>_M$ is lectic ordering in M. Given a set we popped from $\mathcal{Q}$, we test all sets with only 1 more elements and lectically greater. Because we begin by the empty set, we are sure to go over all possible subsets whenever they are to be investigated. Using lectic ordering avoids testing a subset twice. Note that implications of $\mathcal{L}$ are indeed right-closed. Moreover, the conditional statement for adding an implication in $\mathcal{L}$ is useful since an equivalence class can contain only one set. This (closed) set will be by definition minimal, even though the corresponding implication is useless, whence the test.

Algorithm 9: MinimalGeneratorBasis

Input: A context (G, M, I) and a closure operator $''$
Output: $\mathcal{L}$ the basis of minimal generators

```
queue Q ;
L := ∅ ;
push(Q, ∅ ⟶ ∅'') ;
while Q ≠ ∅ do
    A ⟶ B := pop(Q) ;
    if A ≠ B then L := L ∪ {A ⟶ B} ;
    foreach C ⊆ M: (|C| = |A| + 1) ∧ (C >_M A) do
        if (C − c)'' ≠ C'', ∀c ∈ C then
            push(Q, C ⟶ C'') ;
return L;
```

Algorithm ProperBasis (10) computes proper implications of a given context. Again, starting from $\emptyset \longrightarrow \emptyset''$ we cover all possibilities. For all implications of $\mathcal{L}$ we try to build new sets out of its premises and a given attribute. Observe that $D := C'' - C$ is the *"first part"* of saturating C. If C is not closed, then we may refine D by removing elements so that $D = C^\bullet$ after the $D \neq \emptyset$ conditional

Algorithm 10: ProperBasis

Input: A context (G, M, I) and a closure operator $''$
Output: $\mathcal{L}$ the basis of proper implications

$\mathcal{L} := \emptyset \longrightarrow \emptyset''$;
foreach $m \in M$ **do**
  $\mathcal{L}_t = \mathcal{L}$;
  foreach $A \longrightarrow B \in \mathcal{L}$ **do**
    $C := A \cup \{m\}$;
    $D := C'' - C$;
    if $D \neq \emptyset$ **then**
      foreach $\alpha \longrightarrow \beta \in \mathcal{L}_t$ **do**
        if $(A \subset \alpha) \wedge (\alpha - A \subseteq D)$ **then** remove $\alpha \longrightarrow \beta$ from $\mathcal{L}_t$;
      foreach $\alpha \longrightarrow \beta \in \mathcal{L}$ **do**
        if $\alpha \subset C$ **then** $D := D - \beta$;
    $\mathcal{L} := \mathcal{L} \cup \{C \longrightarrow D\}$;
foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  if $(A \cup B \neq \emptyset) \wedge (A \neq B)$ **then** $B := B - A$;
return $\mathcal{L}$;

statement. Especially the second for loop, which removes from D all elements of C'' we would get with another implication than $C \longrightarrow D$.

Before studying results, we may describe random generation. We would like to put the emphasis on the way we randomly generate sets and implications. First, let us focus on set generation. For a set X, we use discrete uniform random distribution on the interval $[0\ ;\ |\,\Sigma\,|]$:

1. determine the size $|X|$ of X by drawing a number out of our distribution,
2. draw again $|X|$ numbers. Because of the interval, we are sure to obtain valid indices of elements to set in X.

Note that an element can be drawn more than once, resulting in effective $|X|$ smaller than the one we got at the beginning. We do not consider this as an issue. Generating theories then is as follows:

1. generate a conclusion randomly,
2. generate a premise. Because we want implications to be informative, we keep as a premise the difference *premise* $-$ *conclusion*.
3. because empty premise is likely to occur several times, resulting in $\emptyset \longrightarrow \Sigma$, we allow for no more than one empty premise.

When $|\,\mathcal{B}\,|$ is fixed, and $|\,\Sigma\,|$ grows, especially when $|\,\Sigma\,| \gg |\,\mathcal{B}\,|$, we may encounter the problem of generating only non-redundant basis, because we are more likely to generate disjoints sets (the attribute set being very large, for few implications) and hence non-redundant implications only. To bypass this problem, we introduce another randomisation based on Armstrong rules [4], [20]. Given some

implications, those rules are an inference system to derive new implications out of others. Some of them are:

- $\mathcal{L} \models A \longrightarrow B$, then $\mathcal{L} \models A \cup C \longrightarrow B$ (augmentation)
- $\mathcal{L} \models A \longrightarrow B$ and $\mathcal{L} \models C \longrightarrow D$, then $\mathcal{L} \models A \cup C \longrightarrow B \cup D$ (union)
- $\mathcal{L} \models A \longrightarrow B$ and $\mathcal{L} \models B \cup C \longrightarrow D$, then $\mathcal{L} \models A \cup C \longrightarrow D$ (pseudo-transitivity)

Therefore, when we generate theories with an increasing attribute sets, given a number of implications $|\mathcal{B}|$, we only generate randomly a part of (by tenth for simplicity) those implications and we use randomly Armstrong rules to generate the remainder. This creates then a certain amount of redundancy even if $|\Sigma| \gg |\mathcal{B}|$. Nevertheless, for the other case around, redundancy is very likely to be built in, since there is much more implications than attributes. In this case we just generate the required number of implications. Ideally, we would like to supply a certain amount of non-redundancy for which we did not do enough research to find a solution.

Note that this method can be discussed and probably improved. We did not investigate further ways of generations since the main task was to test algorithms and not to study implication structures in depth. This anyway is a much interesting question for further work.

As in [8] we use real datasets from the UCI repository. To summarize informations on datasets we use, see table 1. For all of those 35 bases, we ran the 5 minimization procedure we reviewed and implemented. The value recorded is the execution time in seconds for minimizing the given basis. See the results in table 2. All of the algorithms use only Closure, based on the observations of the efficiency of LinClosure in [8].

First, one can observe the wide range of performances. On the one hand, MinCover, DuquenneMinimization and MaierMinimization do not exceed 100 seconds of execution time, while BercziMinimization and AFPMinimization can require up to several hours. The most striking case is for *Breast Wisconsin*, with proper and generators basis. This is probably because of the number of implications in those cases. Anyway, while the two first algorithms require about 30 to 70 seconds for minimization, BerczIMinimization needs roughly one hour and AFP up to ten or eleven hours. This stresses on the intractability of AFP in practice, and possibly the Angluin algorithm when we are prevented from the speed of an oracle.

Regarding the difference between MinCover and DuquenneMinimization, observe that in general, the algorithm by Duquenne is faster on non-minimum bases and slightly slower on already minimum ones (but the minimum (proper) case we will consider after). In both cases, one hypothesis to explain those gaps could be both redundancy elimination and limitation of closure computations in the second loop of DuquenneMinimization. The number of closure computations in MinCover is likely to be constant whatever the case is, even though those operations become lighter with the reduction of the input basis. In DuquenneMinimization however, apart from first left-quasi-closure and redundancy elimination allowing for several savings in closure computations, we

$\mathcal{L}$		$\lvert\Sigma\rvert$	$\lvert\mathcal{B}\rvert$
Zoo	minimum	28	141
	generators		874
	proper		554
Flare	minimum	49	3382
	generators		39787
	proper		10692
Breast cancer	minimum	43	3352
	generators		16137
	proper		11506
Breast Wisconsin	minimum	91	10640
	generators		51118
	proper		45748
Post operative	minimum	26	625
	generators		3044
	proper		1721
SPECT	minimum	23	2169
	generators		44341
	proper		8358
Vote	minimum	18	849
	generators		8367
	proper		2410

Table 1: Summary of real datasets characteristics

stop an iteration of the second loop whenever some quasi-closed premise is not pseudo-closed. This break point permits to omit closure computations and in practice, thanks to lectic ordering, we may not need to go over all implications of the output base to check whether an implication is redundant or not. The only case where the second loop should require more computations is when the base is already minimum, because for each step of the second loop, we have to go over all implications of the growing output system. As remarked in most of execution times, when the basis is already minimum DuquenneMinimization performs somehow slightly worse than MinCover, which could be explained by the previous hypothesis. However, as we mentioned at the beginning of this paragraph, when it comes at minimizing the minimum base we obtained from the proper one, DuquenneMinimization tends to be a bit better. In fact, implications of the proper basis are not right-closed, therefore there is a possibility in the first loop of MinCover (because we use Closure) to run over all implications twice: one loop is sufficient to get the closure, but a second is required by the algorithm to exhibit no updates. And again, because of non-redundancy elimination at this step, DuquenneMinimization may get the point by first removing numerous redundant implications reducing the time spent in closures.

One can also denote that MaierMinimization seems to be as efficient as DuquenneMinimization except in the minimum cases where it performs worse (compared to DuquenneMinimization and MinCover). Our hypothesis is the

$\mathcal{L}$		MinCover	Duquenne	Maier	Berczi	AFP
Zoo	DG	< 0.001	< 0.001	< 0.001	< 0.001	0.016
	min (mingen)	< 0.001	< 0.001	< 0.001	< 0.001	0.016
	min (proper)	< 0.001	< 0.001	< 0.001	< 0.001	0.016
	proper	0.005	0.002	0.003	0.016	0.063
	mingen	0.007	0.004	0.005	0.047	0.094
Flare	DG	0.097	0.117	0.211	27.922	96.178
	min (mingen)	0.134	0.194	0.288	27.750	98.145
	min (proper)	0.200	0.190	0.308	30.063	111.944
	proper	1.684	0.933	0.917	88.375	402.453
	mingen	16.047	7.981	7.576	160.328	2514.610
Breast Cancer	DG	0.089	0.109	0.203	33.047	90.031
	min (mingen)	0.108	0.126	0.243	26.578	89.516
	min (proper)	0.153	0.132	0.258	29.438	105.141
	proper	1.920	0.864	1.014	93.266	429.844
	mingen	2.006	1.277	1.444	102.562	598.172
Breast Wisconsin	DG	0.940	1.208	2.348	1005.750	3109.920
	min (mingen)	1.277	1.533	3.314	949.953	3140.940
	min (proper)	3.121	2.225	5.514	1682.03	5408.98
	proper	67.147	28.053	30.488	3675.910	40521.0
	mingen	35.488	30.023	29.295	2772.980	38310.200
Operative	DG	0.005	0.006	0.011	0.219	0.734
	min (mingen)	0.007	0.007	0.012	0.219	0.719
	min (proper)	0.006	0.007	0.011	0.234	0.640
	proper	0.039	0.020	0.024	0.594	2.422
	mingen	0.078	0.055	0.055	0.813	4.063
SPECT	DG	0.045	0.066	0.108	10.328	22.454
	min (mingen)	0.061	0.080	0.134	8.156	19.438
	min (proper)	0.078	0.070	0.150	8.250	26.980
	proper	0.930	0.394	0.451	51.063	114.564
	mingen	24.077	10.206	10.858	194.875	863.903
Vote	DG	0.006	0.008	0.015	0.484	1.579
	min (mingen)	0.007	0.011	0.016	0.469	1.516
	min (proper)	0.008	0.009	0.016	0.469	1.593
	proper	0.076	0.036	0.049	1.625	6.203
	mingen	0.584	0.298	0.331	4.109	22.875

Table 2: Comparison of the algorithms on real datasets (execution in s)

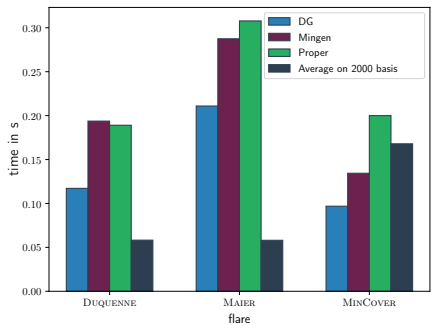
same as for the difference between DuquenneMinimization and MinCover: in fact, the first loop of Maier's algorithm is to get rid of redundant implications, therefore when a basis is highly redundant, first removing them allow to spare numerous closure computations and reduces closure cost. This is not the case in MinCover where we compute right closure of all implications. Once the first step is done in both DuquenneMinimization and MaierMinimization they may perform a similar amount of operations explaining the small gap between them for non-minimum bases:

- DuquenneMinimization performs ordering and then, for all implications, either we need less than $|\mathcal{B}||\Sigma|$ time (because we break the for loop whenever we observe a set is not pseudo-closed) or we will require $|\mathcal{B}||\Sigma| +$ the time for closing since we need to run across $\mathcal{L}_c$ and computing a closure,
- For MaierMinimization, we are sure to perform at least one closure for all implications, to determine equivalence classes. Nevertheless, notice that when removing direct determination, we perform another full closure only if the implication must be kept in the base, otherwise we stop once we found direct determination.

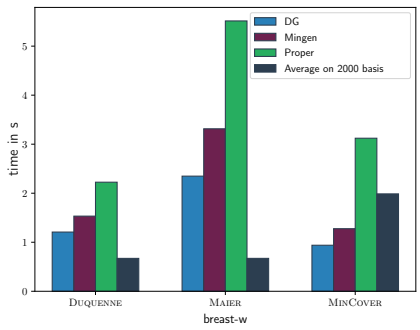
In both case we are likely to perform the same number of set operations if an implication has not to be removed. Still, the difference can rely on which of lectic ordering or determining equivalence classes (which is also a run over a $|\mathcal{B}| \times |\mathcal{B}|$ matrix!). So far, we did not find the true explanations. One possible test to run in further work is using a profiler to see whether the small difference is explained by performance of the computer, or by the structure of the basis we are minimizing.

However, our discussion yields another idea on the worst case of minimization. In fact, the case where we would require more time given fixed parameters $|\Sigma|$ and $|\mathcal{B}|$, is when there is no implication to remove. From our point of view, having no implication to remove maximizes the computations, because all operations of removing redundant implications, figuring out closures and so forth do not reduce the basis. To see whether our idea is interesting or not, we ran few more tests: we took the first three algorithms (because they are faster) and characteristics of the minimum bases. Then, we took the average minimization time over 2000 randomly generated theories of corresponding size. We omit *"zoo"* and *"Breast cancer"* datasets since the first one has very low results and the characteristics of the second one are almost the same as *"Flare"*. Results are presented in figure 1.

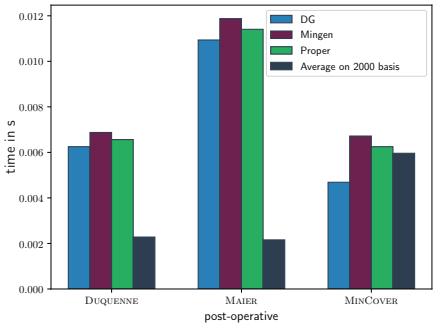
Each graph corresponds to a dataset. Inside all of them, we have a series of bar plots per algorithm. In order, one may find the time (in seconds) required for the Duquenne-Guigues basis, the minimum basis we had with the minimal generators, with proper implications and eventually the average time we obtained for minimization over 2000 randomly generated bases. For DuquenneMinimization and MaierMinimization, it seems like our hypothesis holds: on average, minimizing a redundant system is less time consuming than a base being minimum, for fixed $|\Sigma|$ and $|\mathcal{B}|$. However, this does not hold for MinCover for



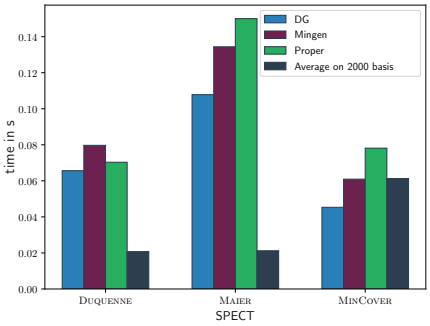
(a) Flare dataset: $|\Sigma| = 49$, $|\mathcal{B}| = 3382$



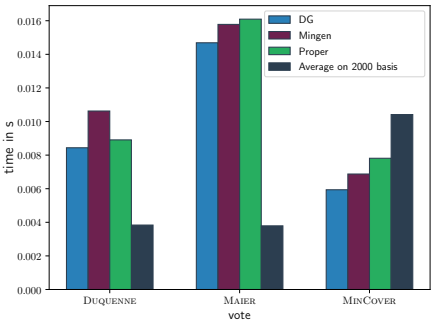
(b) Breast Wisconsin: $|\Sigma| = 91$, $|\mathcal{B}| = 10640$



(c) Post operative: $|\Sigma| = 26$, $|\mathcal{B}| = 625$



(d) SPECT: $|\Sigma| = 23$, $|\mathcal{B}| = 2169$

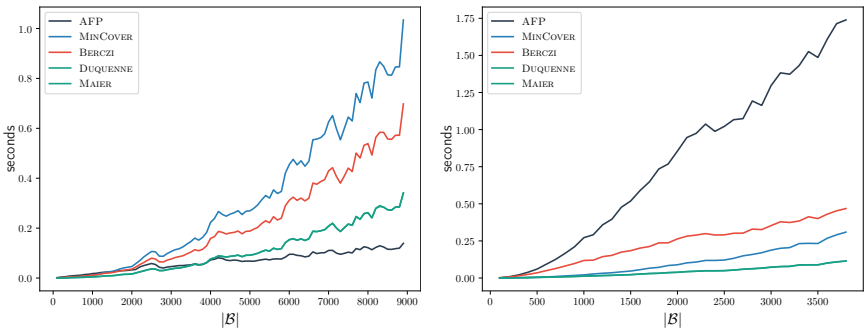


(e) Vote: $|\Sigma| = 18$, $|\mathcal{B}| = 849$

Figure 1: Average minimization against minimal basis

which the worst case is often the minimum bases with non right-closed implications. Moreover reducing a redundant theory is more expensive on average than minimizing right-closed bases. Our idea is that right-closed results in fast closure computations, because for each implication the information we may get is maximal. In the case of non-closed implications, we are likely to need several runs over $\mathcal{L}$ for this task. Then, depending on the order in which we consider implications in the second loop, the execution time may vary. If all redundant implications are handled before others, we will spare several operations. On the contrary, if they are all placed at the end, we will maximize the cost.

Last, we used our random generation tool to try to trace the evolution of each algorithms when one parameter is fixed, namely $|\Sigma|$. One can observe the results in figure 2. We would like to emphasize on the experimental aspect of those results: they act as a hint for further work and not as a ground for hypothesis as previous ones. Observe that in smallest case, AFP is faster than all other algorithms, while in the other one, results coincide with observed real data. It can be due to random generation only exploring one part of the system space being (for instance) highly redundant. It could also be related to the small size of the dataset we are minimizing here. Whether those observation are to be investigated or not is a matter of further study and experiment. Still, they depict interesting track to follow in the future on the behaviour of AFP as much as the underlying structure of our implication systems.



(a) Average time, $|\Sigma| = 100$ (b) Average time, $|\Sigma| = 500$

Figure 2: Random generated tests for fixed $|\Sigma|$ (over 500 ex)

We would like to remind that those results are valid within the scope of our tests and can be somehow related to our way of generating random data for instance. Indeed, we lack insight on whether the systems we randomly generate are a fine representation of the theories space. This highlights, with our previous observations, the interest of studying the underlying structure of the systems

we are trying to minimize: redundancy, minimality, right-closedness and so forth which separate the algorithms we studied in practice. Still, We cannot assume our results as a general truth and they should be tested in further work on other datasets. Plus, one could enquiry about more statistical measures to sharpen the understanding of those procedures. This relativity allows for perspective in testing, trying to increase the number of real datasets, improve random generation

6 Conclusion

In this paper we provided a review of existing algorithms for horn minimization. Navigating in various communities we came up with 5 of them having different complexities and operations. More than explaining their theoretical aspects, we also tried to implement and compare them under practical testing with real data. Those tests indicated that depending on the systems we may find different behaviours. However, it also resulted that redundancy elimination seems to be in practice a step to be considered for efficient reduction. However, investigation of systems structure (hence random generation!), deeper AFP analysis are to be done in further work so that validity of our experiments overtakes the scope of our tests.

References

[1] Aho, A. V., Garey, M. R., and Ullman, J. D. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing* (July 2006).

[2] Angluin, D., Frazier, M., and Pitt, L. Learning conjunctions of Horn clauses. *Machine Learning 9*, 2 (July 1992), 147–164.

[3] Arias, M., and Balcázar, J. L. Canonical Horn Representations and Query Learning. In *Algorithmic Learning Theory* (Berlin, Heidelberg, 2009), R. Gavaldà, G. Lugosi, T. Zeugmann, and S. Zilles, Eds., Springer Berlin Heidelberg, pp. 156–170.

[4] Armstrong, W. W. Dependency Structures of Data Base Relationships. In *IFIP Congress* (1974), pp. 580–583.

[5] Ausiello, G., D'Atri, A., and Saccà, D. Graph Algorithms for Functional Dependency Manipulation. *J. ACM 30*, 4 (Oct. 1983), 752–766.

[6] Ausiello, G., D'Atri, A., and Saccá, D. Minimal Representation of Directed Hypergraphs. *SIAM J. Comput. 15*, 2 (May 1986), 418–431.

[7] Ausiello, G., and Laura, L. Directed hypergraphs: Introduction and fundamental algorithms—A survey. *Theoretical Computer Science 658*, Part B (2017), 293 – 306.

[8] Bazhanov, K., and Obiedkov, S. Optimizations in computing the Duquenne–Guigues basis of implications. *Annals of Mathematics and Artificial Intelligence 70*, 1-2 (Feb. 2014), 5–24.

[9] Beeri, C., and Bernstein, P. A. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst. 4*, 1 (Mar. 1979), 30–59.

[10] Bertet, K., Demko, C., Viaud, J.-F., and Guérin, C. Lattices, closures systems and implication bases: A survey of structural aspects and algorithms. *Theoretical Computer Science* (Nov. 2016).

[11] Boros, E., Čepek, O., and Kogan, A. Horn minimization by iterative decomposition. *Annals of Mathematics and Artificial Intelligence 23*, 3-4 (Nov. 1998), 321–343.

[12] Boros, E., Čepek, O., Kogan, A., and Kučera, P. Exclusive and essential sets of implicates of Boolean functions. *Discrete Applied Mathematics 158*, 2 (2010), 81 – 96.

[13] Boros, E., Čepek, O., and Makino, K. Strong Duality in Horn Minimization. In *Fundamentals of Computation Theory* (Berlin, Heidelberg, 2017), R. Klasing and M. Zeitoun, Eds., Springer Berlin Heidelberg, pp. 123–135.

[14] Bérczi, K., and Bérczi-Kovács, E. R. Directed hypergraphs and Horn minimization. *Information Processing Letters 128* (2017), 32 – 37.

[15] Cori, R., and Lascar, D. *Mathematical Logic: Part 1: Propositional Calculus, Boolean Algebras, Predicate Calculus, Completeness Theorems*. OUP Oxford, Sept. 2000. Google-Books-ID: Cle6_dOLt2IC.

[16] Davey, B. A., and Priestley, H. A. *Introduction to Lattices and Order*. Cambridge University Press, 2002.
[17] Day, A. The Lattice Theory of Functional Dependencies and Normal Decompositions. *International Journal of Algebra and Computation* (1992).
[18] Duquenne, V. Some variations on Alan Day's algorithm for calculating canonical basis of implications. In *Concept Lattices and their Applications (CLA)* (Montpellier, France, 2007), pp. 17–25.
[19] Ganter, B. Two Basic Algorithms in Concept Analysis. In *Formal Concept Analysis* (Mar. 2010), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 312–340.
[20] Ganter, B., and Obiedkov, S. *Conceptual Exploration*. Springer, 2016.
[21] Ganter, B., and Wille, R. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin Heidelberg, 1999.
[22] Guigues, J., and Duquenne, V. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines 95* (1986), 5–18.
[23] Hammer, P. L., and Kogan, A. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence 64*, 1 (Nov. 1993), 131–145.
[24] Maier, D. Minimum Covers in Relational Database Model. *J. ACM 27*, 4 (1980), 664 – 674.
[25] Maier, D. *Theory of Relational Databases*. Computer Science Pr, 1983.
[26] Shock, R. C. Computing the minimum cover of functional dependencies. *Information Processing Letters 22*, 3 (Mar. 1986), 157–159.
[27] Wild, M. Implicational bases for finite closure systems. *Informatik-Bericht 89/3, Institut fuer Informatik* (Jan. 1989).
[28] Wild, M. A Theory of Finite Closure Spaces Based on Implications. *Advances in Mathematics 108*, 1 (Sept. 1994), 118–139.
[29] Wild, M. Computations with finite closure systems and implications. In *Computing and Combinatorics* (Aug. 1995), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 111–120.
[30] Wild, M. The joy of implications, aka pure Horn formulas: Mainly a survey. *Theoretical Computer Science 658* (2017), 264 – 292.