Horn Minimization

Obiedkov Sergei, Vilmin Simon

June 12, 2018

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 provide ideas on which closure operator matches the best each algorithm and 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 [29]. 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 [14]. 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 [12, 6, 19, 5, 29, 10, 22]. 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 [20, 18] for an introduction) and Database Theory (DB) (see [24]). Both sides were developed independently in the early 80's. For the first domain, characterization of

minimality goes to Duquenne and Guigues [21], 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 ([24, 23]). The polynomial time algorithm he gives for minimization heavily relies on a fast subroutine discovered by Beeri and Bernstein in [8], 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 ([25]). Unfortunately, as we shall see and as already discussed by Wild in [28] 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 [6, 4, 5]. 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, [18] 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 [16] to derive an algorithm based on congruence relations. For in-depth knowledge of implication system within lattice terminology, we can see [15] as an introduction and [9] for a survey. Next, Wild ([26, 27, 28]) 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. [16], [25]). This framework is the one we will use for our study, and can be found in more recent work by Ganter & Obiedkov in [19]. Later, Duquenne proposed some variations of Day's work with another algorithm in [17]. More recently, Boròs et al. by working in a boolean logic framework, exhibited a theorem on the size of canonical basis [11, 12]. 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 [13]. Furthermore, an algorithm for computing the canonical basis starting from any system is given in [19]. 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 ([21]). 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 (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.

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 [8] called LinClosure addresses this question. LinClosure as previously mentioned has been widely used, notably in [24, 23, 19, 25, 16]. 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.).

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);
```

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 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*.

Algorithm 2: LinClosure

Input: A base $\mathcal{L}$, $X \subseteq \Sigma$

Output: The closure $\mathcal{L}(X)$ of X under $\mathcal{L}$

```
foreach A ⟶ B ∈ ℒ do
    count[A ⟶ B] := |A| ;
    if |A| = 0 then
        X := X ∪ B ;
    foreach a ∈ A do
        list[a] = list[a] ∪ {A ⟶ B} ;
update := X ;
while update ≠ ∅ do
    choose m ∈ update ;
    update := update − {m} ;
    foreach A ⟶ B ∈ list[m] do
        count[A ⟶ B] := count[A ⟶ B] − 1 ;
        if count[A ⟶ B] = 0 then
            add := B − X ;
            X := X ∪ add ;
            update := update ∪ add ;
return X ;
```

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 [28]. 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 [7]. 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 [16] and by Wild somehow in [26]. It can be found as we will write it in [19]. The principle is to perform

right-saturation first so has to have right-closed implications. Next, it computes left-saturation to find pseudo-closed sets and remove redundant implications. See algorithm 3.

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**
    $\mathcal{L} := \mathcal{L} \cup \{A \longrightarrow B\}$;

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.

The second algorithm we have has been discussed by Duquenne in [17] as variations of Day algorithms. In particular, one can find in DuquenneMinimization (see algorithm 4) an alternative to MinCover.

In this algorithm, we first perform left-saturation 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. It 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 [23, 24]. Wild discussed it and its connection with closure spaces framework in [26]. 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

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 $\mathcal{L}_c$;

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 $\mathcal{L} - E_{\mathcal{L}}[A] \models A \longrightarrow C$. 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 $\exists C \longrightarrow D \in E_{\mathcal{L}}(X)\ s.t\ A \xrightarrow{\mathrm{d}} C$ **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 [13].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.

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]).

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 $\mathcal{L}_c$;

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 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 BercziMinimization 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 BercziMinimization. From this point of view, it gets closer to Angluin algorithm.

5 Experiments

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).

Algorithm 8: AFPMinimization

Input: some theory $\mathcal{L}$ over Σ

Output: $\mathcal{L}_c$ the Duquenne-Guigues basis of $\mathcal{L}$

$\mathcal{L}_c := \emptyset$;
Stack $\mathcal{S}$;
forall $A \longrightarrow B \in \mathcal{L}$ **do**
  $\mathcal{S} := [A]$;
  repeat
    $X := \mathcal{L}_c(\text{pop}(\mathcal{S}))$;
    if $X \neq \mathcal{L}(X)$ **then**
      $found := \bot$;
      forall $\alpha \longrightarrow \beta \in \mathcal{L}_c$ **do**
        $C := \alpha \cap X$;
        if $C \neq \alpha$ **then**
          $D := \mathcal{L}(C)$;
          if $C \neq D$ **then**
            $found := \top$;
            change $\alpha \longrightarrow \beta$ by $C \longrightarrow D$ in $\mathcal{L}_c$;
            push($X \cup D$, $\mathcal{S}$) ;
            if $\beta \neq D$ **then**
              push(α, $\mathcal{S}$);
            exit for
      if $found = \bot$ **then**
        $\mathcal{L}_c := \mathcal{L}_c \cup \{X \longrightarrow \mathcal{L}(X)\}$;
  until $\mathcal{S} = \emptyset$;
return $\mathcal{L}_c$;

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

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

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

[7] 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.

[8] 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.

[9] 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).

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] Davey, B. A., and Priestley, H. A. *Introduction to Lattices and Order.* Cambridge University Press, 2002.

[16] Day, A. The Lattice Theory of Functional Dependencies and Normal Decompositions. *International Journal of Algebra and Computation* (1992).

[17] 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.

[18] 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.

[19] Ganter, B., and Obiedkov, S. *Conceptual Exploration*. Springer, 2016.

[20] Ganter, B., and Wille, R. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin Heidelberg, 1999.

[21] 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.

[22] Hammer, P. L., and Kogan, A. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence 64*, 1 (Nov. 1993), 131–145.

[23] Maier, D. Minimum Covers in Relational Database Model. *J. ACM 27*, 4 (1980), 664 – 674.

[24] Maier, D. *Theory of Relational Databases*. Computer Science Pr, 1983.

[25] Shock, R. C. Computing the minimum cover of functional dependencies. *Information Processing Letters 22*, 3 (Mar. 1986), 157–159.

[26] Wild, M. Implicational bases for finite closure systems. *Informatik-Bericht 89/3, Institut fuer Informatik* (Jan. 1989).

[27] Wild, M. A Theory of Finite Closure Spaces Based on Implications. *Advances in Mathematics 108*, 1 (Sept. 1994), 118–139.

[28] 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.

[29] Wild, M. The joy of implications, aka pure Horn formulas: Mainly a survey. *Theoretical Computer Science 658* (2017), 264 – 292.