On the Computation of a Productive Partially Ordered Possibilistic Repair

Ahmed Laouar[1], Sihem Belabbes[2] and Salem Benferhat[1]

[1]*CRIL, Univ. Artois & CNRS, UMR 8188, Lens, 62300, France*
[2]*LIASD, IUT de Montreuil, Univ. Paris 8, Saint-Denis, 93200, France*

Abstract

We deal with repairing inconsistent partially ordered lightweight ontologies in the possibilistic setting. More specifically, we consider the closure-based Cπ-repair method, which yields a more productive partially ordered possibilistic repair and is tractable in DL-Lite$_\mathcal{R}$. In this work, we refine the characterization of the Cπ-repair method and propose an equivalent algorithm that is more efficient. We illustrate our findings with an experimental analysis. In particular, we highlight the main situations in which the Cπ-repair method achieves the best performance both in terms of productivity and computational cost.

Keywords

Lightweight Ontologies, Inconsistency, Partial orders, Data repairs

1. Introduction

Reasoning with inconsistent lightweight ontologies commonly consists in evaluating queries over the repairs of the ABox, defined as maximal subsets of the ABox that are consistent with respect to the TBox. Inconsistency-tolerant semantics are strategies for selecting which repairs to query in order to derive valid conclusions from an inconsistent ontology. Some of the most prominent semantics have been implemented in reasoning systems. For example, the CQAPri system (Consistent Query Answering with Priorities) [1] resolves conjunctive queries in DL-Lite$_\mathcal{R}$ ontologies over the repairs of the ABox. It returns as valid conclusions the query answers that follow: from every repair under the ABox Repair (AR) semantics [2], from the intersection of all the repairs under the Intersection of ABox Repair (IAR) semantics [2], and from any repair under the brave semantics [3]. Another example is the QuID system [4] which implements conjunctive query answering under the IAR semantics in ontologies specified in DL-Lite$_\mathcal{A}$.

The issue of inconsistency management in DL-Lite$_\mathcal{R}$ has been investigated for both totally ordered ontologies [1, 5] and partially ordered ontologies [6]. Some other methods focus on a preference relation defined only over minimal inconsistent subsets of the ABox. The issue was first investigated in prioritized databases [7] and then adapted to DL-Lite in [8].

Besides, inconsistency has been considered within the framework of possibility theory. One can cite the method for computing a totally ordered possibilistic repair [9], which infers all the assertions that are strictly more certain than some inconsistency degree of the ABox. The possibilistic repair is tractable in DL-Lite$_\mathcal{R}$, because its computation can be reduced to checking the consistency of a subset of the ABox. This makes it more efficient than the other repairs that apply to totally ordered ABoxes [9].

Furthermore, in the case of partially ordered ontologies, there is the method for computing the partially ordered possibilistic repair (π-repair) [10], and its closure-based counterpart (Cπ-repair) [11], which is more productive. In both methods, the partial order over the ABox is interpreted as a family of total orders which define the compatible totally ordered ABoxes extending the partial order. The π-repair is obtained from the intersection of the totally ordered possibilistic repairs of the compatible ABoxes, whereas the Cπ-repair intersects their closure. Note that the π-repair is included in the Cπ-repair.

Equivalent characterizations have been proposed for the π-repair and the $C\pi$-repair methods, without exhibiting the all compatible ABoxes, and have been shown to be tractable in DL-Lite$_\mathcal{R}$. The π-repair method is characterized by the notion of π-accepted assertion [12], which boils down to checking the consistency of the said assertion together with the subset of all the assertions that are at least equally certain or are incomparable to it. The $C\pi$-repair method is characterized using the notions of support and dominance [11] against the conflicts of the ABox, which are minimal subsets of the ABox that are inconsistent with respect to the TBox. Basically, a valid conclusion is an assertion that can be derived from consistent minimal subsets of the ABox, called supports, that dominate (i.e., are strictly more certain than) all the conflicts.

In this work, we first implement a naive algorithm for the $C\pi$-repair method, using the tractable characterization based on the notions of support and dominance. We then provide an improved equivalent algorithm by exploiting the fact that $C\pi$-repair is composed of the π-repair and a complement set. We revise the notion of dominance by exploiting the partial order relation in order to reduce the number of conflicts and supports that need to be processed by the algorithm. Thus, computing $C\pi$-repair amounts to computing the π-repair with the tractable characterization of π-accepted assertions, and computing the rest by applying the revised notion of dominance to the remaining assertions. We perform an experimental analysis and confirm that computing the $C\pi$-repair with the new algorithm benefits from the efficiency of computing the π-repair. We highlight the main situations in which the $C\pi$-repair method achieves the best performance both in terms of productivity and computational cost.

This paper is structured as follows. Section 2 recalls some preliminaries then describes a naive algorithm for the $C\pi$-repair method. Section 3 studies properties of $C\pi$-repair which serve to introduce an improved algorithm. Section 4 provides an experimental evaluation of these algorithms. A brief discussion concludes the paper.

2. Preliminaries: The Cπ-repair method

DL-Lite$_\mathcal{R}$ Syntax   A DL-Lite$_\mathcal{R}$ ontology [13] is a finite knowledge base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, built by recursively applying the following grammar:
$$R := P \mid P^- \qquad E := R \mid \neg R \qquad B := A \mid \exists R \qquad C := B \mid \neg B$$
where A is a concept name, P is a role name and P^- is its inverse. The symbol $\neg$ designates a complement set and $\exists$ denotes existential restriction on roles.
The TBox $\mathcal{T}$ contains axioms of the form $B \sqsubseteq C$ and $R \sqsubseteq E$. The ABox $\mathcal{A}$ contains assertions of the form $A(a)$ and $P(a, b)$, where a and b are individuals. The axioms in $\mathcal{T}$ may be positive inclusions of the form $B_1 \sqsubseteq B_2$ or $R_1 \sqsubseteq R_2$ which allow to derive new assertions from $\mathcal{A}$. The axioms in $\mathcal{T}$ may also be negative inclusions of the form $B_1 \sqsubseteq \neg B_2$ or $R_1 \sqsubseteq \neg R_2$ which serve to exhibit the conflicts in $\mathcal{A}$. A conflict is a minimal subset of $\mathcal{A}$ that is inconsistent with respect to $\mathcal{T}$, with a size of (at most) two assertions in DL-Lite$_\mathcal{R}$ [14]. Inconsistency refers to the absence of a model for the KB. We omit the semantics of DL-Lite$_\mathcal{R}$ for space considerations.

Partially preordered KB   A partial preorder $\unrhd$ over an ABox $\mathcal{A}$ is a reflexive and transitive binary relation. Let $\rhd$ be the associated strict order. Let $\bowtie$ denote incomparability, i.e., for φ_j and φ_k two assertions, $\varphi_j \bowtie \varphi_k$ means that neither $\varphi_j \unrhd \varphi_k$ nor $\varphi_k \unrhd \varphi_j$ applies. The partially preordered ABox is denoted by $\mathcal{A}_{\unrhd}$. In the rest of this paper, we deal with an inconsistent partially preordered KB and denote it by $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_{\unrhd} \rangle$.

Next, we provide an algorithmic definition for each of the core notions of the $C\pi$-repair method, namely the deductive closure, the conflict set and the support of an assertion.

2.1. Deductive closure of the ABox

The deductive closure of the ABox contains all the assertions that may be inferred from the KB using the positive axioms of the TBox. The individuals included in the closure are limited to those present

in the ABox, which makes the closure finite. Algorithm 1 computes the deductive closure of $\mathcal{A}_\rhd$ by running a set of First Order SQL queries. For each concept or role name in an axiom of the TBox, it creates a conjunctive query (CQ) $q(\overrightarrow{x})$ to retrieve the individuals in the corresponding assertion (in line 3). Then, the algorithm reformulates the query into a union of conjunctive queries (UCQ), under classical semantics, to get all the individuals present in any subsume or subrole of the initial concept or role name. Any CQ-rewriting procedure may be used. Here, we use the PerfectRef algorithm [13]. The results of the reformulation are then executed on the ABox (in line 6). This corresponds to computing the set of the answers for each query over $\mathcal{I}_{\mathcal{A}_\rhd}$, which is the interpretation that satisfies exactly the assertions of $\mathcal{A}_\rhd$.

Algorithm 1: ComputeClosure

Input: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_\rhd \rangle$: a KB
Output: $cl(\mathcal{A}_\rhd)$: the deductive closure of $\mathcal{A}_\rhd$.

1 $cl(\mathcal{A}_\rhd) \leftarrow \emptyset$
2 **foreach** N : *a concept name or role name* **in** $\mathcal{T}$ **do**
3     $q(\overrightarrow{x}) \leftarrow \delta(\mathsf{N})$
4   PR $\leftarrow$ PerfectRef$(q, \mathcal{T})$
5   **foreach** $q_i(\overrightarrow{x})$ **in** PR **do**
6   **foreach** $\overrightarrow{a} \in ans(q_i, \mathcal{I}_{\mathcal{A}_\rhd})$ **do**
7           $cl(\mathcal{A}_\rhd) \leftarrow cl(\mathcal{A}_\rhd) \cup \{\lambda(\mathsf{N}, \overrightarrow{a})\}$

8 **return** $(cl(\mathcal{A}_\rhd))$

- δ: a translation function mapping a concept name B with a variable x, and a role name R with the variables x_1, x_2 (adapted from [13]).

- ans$(q_i, \mathcal{I}_{\mathcal{A}_\rhd})$: the result of running the query q_i on the interpretation $\mathcal{I}_{\mathcal{A}_\rhd}$.

- λ: a translation function mapping the concept name or the role name N with each of the answers $\overrightarrow{a}$ (adapted from [13]).

2.2. Conflict set of the ABox

In an inconsistent DL-Lite$_\mathcal{R}$ KB, a conflict is a minimal subset of (at most) two assertions that are contradictory in terms of the axioms [14]. In the following, we assume that a conflict contains exactly two assertions, and denote the conflict set of the ABox $\mathcal{A}_\rhd$ by $\mathsf{Cf}(\mathcal{A}_\rhd)$. Algorithm 2 uses query generation and reformulation (similar to the algorithm in [15]) to compute the conflict set. The algorithm creates a conjunctive query $(q_i(\overrightarrow{x}))$ for each negative axiom in $\mathcal{T}$. Then it performs CQ-rewriting under classical semantics, to get queries for the negative axioms that are not explicitly stated in the TBox. The size of each query at this step is 2. After that, the obtained queries are evaluated over the ABox to get the individuals $(\overrightarrow{a})$ that are present in both assertions of any conflicting concept names or role names. $\overrightarrow{a}$ are substituted for the variables $\overrightarrow{x}$, and the set of atoms in $q_i(\overrightarrow{a})$ are added to the conflict set.

Algorithm 2: ComputeConflicts

Input: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_\rhd \rangle$: a KB
Output: $\mathsf{Cf}(\mathcal{A}_\rhd)$: the conflict set of $\mathcal{A}_\rhd$

1 $\mathsf{Cf}(\mathcal{A}_\rhd) \leftarrow \emptyset$
2 $Q \leftarrow \emptyset$
3 **foreach** α: *a negative axiom* **in** $\mathcal{T}$ **do**
4     $Q \leftarrow Q \cup \{q_i(\overrightarrow{x}) \leftarrow \delta(\alpha)\}$
5 PR $\leftarrow \bigcup_{q_i \in Q}$ PerfectRef$(q_i, \mathcal{T})$
6 **foreach** $q_i(\overrightarrow{x})$ **in** PR **do**
7   **foreach** $\overrightarrow{a} \in ans(q_i, \mathcal{I}_{\mathcal{A}_\rhd})$ **do**
8        $\mathsf{Cf}(\mathcal{A}_\rhd) \leftarrow \mathsf{Cf}(\mathcal{A}_\rhd) \cup \{\{atoms(q_i(\overrightarrow{a}))\}\}$

9 **return** $(\mathsf{Cf}(\mathcal{A}_\rhd))$

- δ: a translation function from a negative axiom α in $\mathcal{T}$ to a conjunctive query (defined in [13]).

- ans$(q_i, \mathcal{I}_{\mathcal{A}_\rhd})$: the result of running the query q_i on the interpretation $\mathcal{I}_{\mathcal{A}_\rhd}$.

- atoms$(q_i(\overrightarrow{a}))$: the set of atoms in the query $q_i(\overrightarrow{a})$.

2.3. Supports of an assertion

The support of an assertion in an ABox is a minimal consistent subset of the ABox that allows to derive it. In DL-Lite$_\mathcal{R}$, a support consists of at most a single assertion of the ABox. Algorithm 3 computes

the union of boolean queries that represent the instance checking queries of the supports for a given assertion. This is achieved via the reformulation of the instance checking query of the assertion under classical semantics (using the PerfectRef algorithm). The verified instances are added to the set of supports of the assertion.

Algorithm 3: ComputeSupports

Input: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_{\rhd} \rangle$: a KB, $B(a)$, resp. $R(a_1, a_2)$: an assertion in $cl(\mathcal{A}_{\rhd})$
Output: $\mathcal{S}$: the supports of $B(a)$, resp. $R(a_1, a_2)$, in $\mathcal{A}_{\rhd}$

1   $\mathcal{S} \leftarrow \emptyset$
2   $q(\overrightarrow{a}) \leftarrow B(a)$ /* resp. $R(a_1, a_2)$ */
3   $\mathsf{PR} \leftarrow \mathsf{PerfectRef}(q, \mathcal{T})$
4   **foreach** $q_i(\overrightarrow{a})$ **in** PR **do**
5   **if** $ans(q_i, \mathcal{I}_{\mathcal{A}_{\rhd}})$ **is** $true$ **then**
6            $\mathcal{S} \leftarrow \mathcal{S} \cup \{\{\mathsf{atoms}(q_i(\overrightarrow{a}))\}\}$

7   **return** $(\mathcal{S})$

2.4. Cπ-repair algorithm

Given the notions of deductive closure, conflict set and support introduced in Algorithms 1 to 3, the Cπ-repair method also relies on the notion of dominance. This notion is a way for extending the partial preorder defined over an ABox into a partial preorder defined over subsets of the ABox $\mathcal{A}_{\rhd}$. The dominance is defined as follows.

Definition 1 (Dominance). *Let two subsets $\mathcal{B}_1$ and $\mathcal{B}_2$ of $\mathcal{A}_{\rhd}$. We say that $\mathcal{B}_1$ dominates $\mathcal{B}_2$ if $\forall \varphi_j \in \mathcal{B}_1$, $\exists \varphi_k \in \mathcal{B}_2$ such that $\varphi_j \rhd \varphi_k$.*

The Cπ-repair method is characterized as the dominance of the supports of an assertion over the conflicts of the ABox. The characterization solely applies when the KB is inconsistent. Otherwise, The Cπ-repair amounts to the deductive closure of the ABox.

Definition 2 (Cπ-repair). *A given assertion φ is in $c\pi(\mathcal{A}_{\rhd})$ if $\forall \mathcal{C} \in \mathsf{Cf}(\mathcal{A}_{\rhd})$, $\exists \mathcal{B} \subseteq \mathcal{A}_{\rhd}$ such that $\mathcal{B}$ supports φ (as per Algorithm 3), and $\mathcal{B}$ dominates $\mathcal{C}$ (as per Definition 1).*

Note that in DL-Lite$_{\mathcal{R}}$, a support (a singleton) dominates a conflict (a pair) means that there is an assertion in the ABox that is strictly more certain than at least one element of the conflict.

     Algorithm 4 (given in the next page) is a naive implementation of the Cπ-repair method which applies the characterization given in Definition 2 to each assertion of the closed ABox returned by Algorithm 1.

Lemma 1. *The algorithm Compute$C\pi$-repair (Algorithm 4) runs in polynomial time and space in $|\mathcal{A}_{\rhd}|$ (in data complexity).*

The proofs of all propositions and lemmas are provided in the appendix.

3. Revisiting the Cπ-repair method

In this section, we aim at computing Cπ-repair more efficiently. Thus, we propose to improve Algorithm 4 by exploring two ideas. First, we exhibit additional properties of the dominance relation, in order to reduce the number of the processed conflicts and supports. We do so by focusing on subsets of conflicts and of supports called dominant conflicts and dominant supports. Second, since the Cπ-repair contains the π-repair, we start by computing the π-repair using its tractable characterization. We then complete the Cπ-repair by applying the revised characterization to the assertions that cannot be inferred from the π-repair.

Algorithm 4: ComputeCπ-repair

Input: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_{\rhd} \rangle$: a KB
Output: $C\pi(\mathcal{A}_{\rhd})$: the Cπ-repair of $\mathcal{A}_{\rhd}$

1   $cl(\mathcal{A}_{\rhd}) \leftarrow$ ComputeClosure$(\mathcal{K})$
2   $\mathrm{Cf}(\mathcal{A}_{\rhd}) \leftarrow$ ComputeConflicts$(\mathcal{K})$
3   **if** $\mathrm{Cf}(\mathcal{A}_{\rhd}) = \emptyset$ **then**
4      $\lfloor$ **return** $cl(\mathcal{A}_{\rhd})$
5   **else**
6      $C\pi(\mathcal{A}_{\rhd}) \leftarrow \emptyset$
7   **foreach** φ **in** $cl(\mathcal{A}_{\rhd})$ **do**
8         $\mathcal{S}(\varphi) \leftarrow$ ComputeSupports$(\varphi, \mathcal{K})$
9   visited $\leftarrow \emptyset$ /* visited: the set of visited conflicts */
10   **repeat**
11   **select** $\mathcal{C} = \{c_i, c_j\}$ **from** $\mathrm{Cf}(\mathcal{A}_{\rhd})$
12           $\mathrm{Cf}(\mathcal{A}_{\rhd}) \leftarrow \mathrm{Cf}(\mathcal{A}_{\rhd}) \setminus \{\mathcal{C}\}$
13   visited $\leftarrow$ visited $\cup \{\mathcal{C}\}$
14   supported $\leftarrow$ false   /* supported: a boolean, true if φ has a support */
15   **repeat**
16   **select** $\{s\}$ **from** $\mathcal{S}(\varphi)$
17              $\mathcal{S}(\varphi) \leftarrow \mathcal{S}(\varphi) \setminus \{s\}$
18   **if** $s \rhd c_i$ **or** $s \rhd c_j$ **then**
19                 $\lfloor$ supported $\leftarrow$ true
20   **until** supported **is** true **or** $\mathcal{S}(\varphi) = \emptyset$ /* exit when a support is found or there are no supports */
21   **until** supported **is** false **or** $\mathrm{Cf}(\mathcal{A}_{\rhd}) = \emptyset$ /* exit after parsing all the conflicts or there is no support */
22   **if** supported **is** true **then**
23           $\lfloor$ $C\pi(\mathcal{A}_{\rhd}) \leftarrow C\pi(\mathcal{A}_{\rhd}) \cup \{\varphi\}$ /* φ is supported against each conflict */
24      $\mathrm{Cf}(\mathcal{A}_{\rhd}) \leftarrow \mathrm{Cf}(\mathcal{A}_{\rhd}) \cup$ visited

25 **return** $(C\pi(\mathcal{A}_{\rhd}))$

3.1. Dominant conflicts

Definition 3 (Dominant conflicts). *The set of dominant conflicts, denoted* $\mathrm{Cf}_{dom}(\mathcal{A}_{\rhd})$, *is the maximal subset* $\mathrm{Cf}_{dom}(\mathcal{A}_{\rhd}) \subseteq \mathrm{Cf}(\mathcal{A}_{\rhd})$ *such that* $\forall \mathcal{C}_i \in \mathrm{Cf}(\mathcal{A}_{\rhd})$, *if* $\exists \mathcal{C}_j \in \mathrm{Cf}(\mathcal{A}_{\rhd})$ *such that* $\mathcal{C}_j$ *dominates* $\mathcal{C}_i$, *then* $\mathcal{C}_i \notin \mathrm{Cf}_{dom}(\mathcal{A}_{\rhd})$.

The following proposition states that it is sufficient to consider only the dominant conflicts within the conflict set to compute the Cπ-repair. This stems from the fact that the dominance relation is transitive, which follows from the transitivity of the partial preorder relation $\rhd$.

Proposition 1. *A given assertion* φ *is in* $c\pi(\mathcal{A}_{\rhd})$ *if and only if* $\forall \mathcal{C} \in \mathrm{Cf}_{dom}(\mathcal{A}_{\rhd})$, $\exists \mathcal{B} \subseteq \mathcal{A}_{\rhd}$ *such that* $\mathcal{B}$ *supports* φ *and* $\mathcal{B}$ *dominates* $\mathcal{C}$.

3.2. Dominant supports

Definition 4 (Dominant supports). *Let* $\mathcal{S}(\varphi)$ *be the set of supports of an assertion* φ *in* $\mathcal{A}_{\rhd}$. *The set of dominant supports of* φ, *denoted* $\mathcal{S}_{dom}(\varphi)$, *is the maximal subset* $\mathcal{S}_{dom}(\varphi) \subseteq \mathcal{S}(\varphi)$ *such that* $\forall \mathcal{B}_i \in \mathcal{S}(\varphi)$, *if* $\exists \mathcal{B}_j \in \mathcal{S}(\varphi)$ *such that* $\mathcal{B}_j$ *dominates* $\mathcal{B}_i$, *then* $\mathcal{B}_i \notin \mathcal{S}_{dom}(\varphi)$.

The following proposition states that it is sufficient to consider only the set of dominant supports of any given assertion φ in order to check its membership in Cπ-repair. This result also follows from the transitivity of the partial preorder relation $\rhd$.

Proposition 2. *Let* φ *be an assertion and let* $\mathcal{S}(\varphi)$ *be its set of supports in* $\mathcal{A}_{\rhd}$. φ *is in* $c\pi(\mathcal{A}_{\rhd})$ *if and only if* $\forall \mathcal{C} \in \mathrm{Cf}_{dom}(\mathcal{A}_{\rhd})$, $\exists \mathcal{B} \in \mathcal{S}_{dom}(\varphi)$ *such that* $\mathcal{B}$ *dominates* $\mathcal{C}$.

Proposition 2 entails that Algorithm 4 can actually consider (strict) subsets of the conflicts and the supports. This has the potential for considerably reducing the number of elements over which the algorithm iterates to check membership of an assertion in the Cπ-repair.

3.3. Properties of the Cπ-repair

We exploit two key properties of the Cπ-repair [11] to improve Algorithm 4. First, the Cπ-repair (strictly) contains all the assertions of the π-repair [12], which is characterized as follows.

Definition 5 (π-repair). *Let $\varphi_i \in \mathcal{A}_{\unrhd}$ and $\Delta(\varphi_i) = \mathcal{A}_{\unrhd} \setminus \{\varphi_j | \varphi_j \in \mathcal{A}_{\unrhd}$ such that $\varphi_i \rhd \varphi_j\}$ be the set of assertions that are at least equally certain or incomparable to φ_i. Then $\varphi_i \in \pi(\mathcal{A}_{\unrhd})$ if $\langle \mathcal{T}, \{\varphi_i\} \cup \Delta(\varphi_i) \rangle$ is consistent.*

We conclude that Algorithm 4 does not need to consider all the assertions of the deductively closed ABox in order to compute the Cπ-repair. Instead, an improved version of the algorithm first computes the π-repair [1], before iterating over the remaining assertions of the closed ABox.

Moreover, we explore a second property of the Cπ-repair in order to reduce the number of remaining assertions that need to be checked by the algorithm. This relates to the set difference between the Cπ-repair and the deductive closure of the π-repair. In essence, if an assertion is neither accepted in the π-repair nor its closure, it must have strictly more than one support to be in the Cπ-repair. This is formalized in the following lemma.

Lemma 2. *Let φ be an assertion in $c\pi(\mathcal{A}_{\unrhd})$. Let $\mathcal{S}(\varphi)$ be the set of supports of φ. If $|\mathcal{S}(\varphi)| = 1$, then $\varphi \in cl(\pi(\mathcal{A}_{\unrhd}))$.*

The converse of Lemma 2 does not hold, a non-accepted assertion may have multiple supports.

Algorithm 5 (given in the next page) constitutes an improved version of Algorithm 4. It begins by computing the closed ABox followed by the set of dominant conflicts. If the KB is consistent, the closed ABox is returned. Otherwise, the algorithm computes the assertions of the π-repair and inserts them in the Cπ-repair. Next, it iterates over the assertions of the closed ABox minus the π-repair, and for each such assertion, it computes its set of dominant supports. If one of the supports is in the π-repair, the assertion is added to the Cπ-repair. This step retrieves the assertions that are in the closure of the π-repair. Otherwise, the algorithm checks whether the assertion has more than one support. If it does, then for each conflict, the algorithm checks whether the assertion has a support dominating the conflict (like in Algorithm 4).

Lemma 3. *The algorithm* Compute*Cπ-repair (Algorithm 5) runs in polynomial time and space in* $|\mathcal{A}_{\unrhd}|$ *(in data complexity).*

4. Experimental evaluation

We implemented all the algorithms of this paper in Python. Our system relies on the RDFLib library [16] to read the ontology, a SQLite3 relational DBMS [17] to store the ABox, and the Rapid query rewriting engine [18] to reformulate queries. Our system is publicly available at: https://github.com/ahmedlaouar/py_reasoner. We performed the experiments on an Intel 3.60GHz CPU with 32 GB RAM under Ubuntu, and provided instructions for reproducing them. Let us first discuss how to elicit partial orders to obtain a partially ordered KB.

4.1. Directed acyclic graphs for partial orders

One way for implementing a partially ordered KB is to assign partially ordered priority degrees to the assertions of the ABox [10]. These degrees can be represented graphically using Hasse diagrams or directed acyclic graphs (DAGs). We opt for transitive DAGs [19], which explicitly represent arcs within the graph. In a transitive DAG, nodes represent the degrees of a partially ordered set (POS), an arc indicates the strict preference of a degree over the other, and the absence of an arc expresses incompatibility of two degrees. A set of random DAGs was generated using the R package bnlearn [20], based on different numbers of nodes ($\{50, 100, 500, 1000, 2500\}$) and different probabilities for the

---

[1]A suggested algorithm for computing the π-repair is given in the appendix.

Algorithm 5: ComputeCπ-repair

Input: $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_{\rhd} \rangle$: a KB
Output: $C\pi(\mathcal{A}_{\rhd})$: the Cπ-repair of $\mathcal{A}_{\rhd}$

1 $cl(\mathcal{A}_{\rhd}) \leftarrow$ ComputeClosure($\mathcal{K}$)
2 $\mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) \leftarrow$ ComputeDominantConflicts($\mathcal{K}$)   /* the set given by Definition 3 */
3 **if** $\mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) = \emptyset$ **then**
4   **return** $cl(\mathcal{A}_{\rhd})$
5 **else**
6      $\pi(\mathcal{A}_{\rhd}) \leftarrow$ Computeπ-repair($\mathcal{K}$)
7      $cl(\mathcal{A}_{\rhd}) \leftarrow cl(\mathcal{A}_{\rhd}) \setminus \pi(\mathcal{A}_{\rhd})$
8      $C\pi(\mathcal{A}_{\rhd}) \leftarrow \pi(\mathcal{A}_{\rhd})$
9   **foreach** φ **in** $cl(\mathcal{A}_{\rhd})$ **do**
10        $\mathcal{S}_{dom}(\varphi) \leftarrow$ ComputeDominantSupports($\varphi, \mathcal{K}$)   /* the set given by Definition 4 */
11   **if** $\mathcal{S}_{dom}(\varphi) \cap \pi(\mathcal{A}_{\rhd}) \neq \emptyset$ **then**
12          $C\pi(\mathcal{A}_{\rhd}) \leftarrow C\pi(\mathcal{A}_{\rhd}) \cup \{\varphi\}$
13   **else**
14   **if** $size\,(\mathcal{S}_{dom}(\varphi)) > 1$ **then**
15   visited $\leftarrow \emptyset$ /* visited: the set of visited conflicts */
16   **repeat**
17   **select** $\mathcal{C} = \{c_i, c_j\}$ **from** $\mathsf{Cf}_{dom}(\mathcal{A}_{\rhd})$
18                $\mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) \leftarrow \mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) \setminus \{\mathcal{C}\}$
19   visited $\leftarrow$ visited $\cup \{\mathcal{C}\}$
20   supported $\leftarrow$ false   /* supported: a boolean, true if φ has a support */
21   **repeat**
22   **select** $\{s\}$ **from** $\mathcal{S}_{dom}(\varphi)$
23                  $\mathcal{S}_{dom}(\varphi) \leftarrow \mathcal{S}_{dom}(\varphi) \setminus \{s\}$
24   **if** $s \rhd c_i$ **or** $s \rhd c_j$ **then**
25   supported $\leftarrow$ true
26   **until** supported **is** true **or** $\mathcal{S}_{dom}(\varphi) = \emptyset$ /* exit when support is found or no support */
27   **until** supported **is** false **or** $\mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) = \emptyset$ /* exit if all conflicts parsed or no support */
28   **if** supported **is** true **then**
29                $C\pi(\mathcal{A}_{\rhd}) \leftarrow C\pi(\mathcal{A}_{\rhd}) \cup \{\varphi\}$ /* φ is supported against each conflict */
30             $\mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) \leftarrow \mathsf{Cf}_{dom}(\mathcal{A}_{\rhd}) \cup$ visited

31 **return** $(C\pi(\mathcal{A}_{\rhd}))$

---

presence of arcs $(0.1, \ldots, 0.9)$. A probability indicates the density of the DAG, hence a less dense DAG has more occurrences of incomparability.

4.2. Experimental setting

We used an OWL ontology as the TBox. We opted for the DL-Lite$_{\mathcal{R}}$ version of the modified LUBM benchmark (LUBM$_{20}^{\exists}$) [21] (available at https://home.uni-leipzig.de/clu/). Since this version does not contain negative axioms, we added a set of negative axioms to the TBox in order to create conflicts. Next, we used the Extended University Data Generator (EUDG) [21] to generate three ABoxes of different sizes, and transformed each ABox into a SQLite database. Then, we introduced inconsistency by adding contradictory assertions in each ABox as follows. For each negative axiom inferred from the TBox, we contradict the presence of an individual in a concept assertion (resp. role assertion) with probability p (resp. $\frac{p}{2}$). We used different values of p in order to create five different situations within each ABox in terms of inconsistency, which is measured with by the number of conflicts. For each execution of the algorithms, degrees from a previously generated DAG are randomly assigned to the assertions of each ABox. We refer to the π-repair (Definition 5) and Algorithms 4 and 5 by π, $c\pi$ and $c\pi$++, respectively. We consider three different cases for the evaluation:

Case 1: We analyse both the performance of the algorithms as a function of the size of the ABox and the

ID	#A	# Cf	Cf (s)	#π	π (s)	#$c\pi$	$c\pi$ (s)	$c\pi$++ (s)
u.5p5e-6	9156	111	1.08	8078	1.24	22740	27.65	11.26
u.5p1e-5	9158	387	1.42	5344	1.33	15859	30.56	18.15
u.5p5e-5	9193	1268	1.43	742	1.28	2473	30.15	30.86
u.5p5e-4	9228	3372	1.6	1102	1.41	3651	30.75	29.35
u.5p1e-3	9375	7733	1.7	442	1.57	1475	28.83	28.65
u1p5e-6	75671	753	2.51	36083	4.36	66867	203.59	53.45
u1p1e-5	75678	3049	2.77	22804	8.38	46257	261.47	74.74
u1p5e-5	75718	10388	4.14	5856	5.07	14992	267.33	130.9
u1p15e-5	75842	28553	4.84	2378	6.54	6834	258.71	168.25
u1p5e-4	76255	83863	6.5	988	6.9	2994	251.23	170.61
u5p5e-6	463349	4505	22.48	74511	39.39	167473	6635.11	2160.48
u5p1e-5	463400	8327	23.6	30204	38.89	78857	6711.89	2941.02
u5p5e-5	463684	57054	26.3	8693	35.85	25921	5772.07	3105.82
u5p1e-4	464060	121621	29.76	6213	34.12	18794	6492.42	3566.24
u5p5e-4	466785	544500	49.95	1470	95.88	4713	5868.32	3956.46

Table 1
ABoxes in terms of: size (#**A**), conflicts number (# **Cf**) and computation time (**Cf (s)**), π-repair size (#π) and computation time (**π (s)**), Cπ-repair size (#$c\pi$) and computation time (**$c\pi$ (s)** and **$c\pi$++ (s)**).

impact of the number of conflicts on the size of the repairs and the running time (for computing the repairs). The results are presented in Table 1 and Figure 1.

Case 2: We consider different configurations for the associated DAG (POS). We analyze the impact of changing the size and density of the DAG. This allows us to measure the difference $c\pi(\mathcal{A}_{\triangleright}) \setminus cl(\pi(\mathcal{A}_{\triangleright}))$. We point out the main situations where the Cπ-repair is more productive than the π-repair. The results are depicted in Figure 2 [left].

Case 3: We use ABox **u.5** to compare the proportion of time spent in each step of Algorithms 4 and 5. For each ABox by number of conflicts, we computed the median time from different executions. The results are presented in Figure 2 [middle] and [right].

4.3. Experimental results

Impact of the number of conflicts (Case 1)   First, we compare the running time of Algorithms 4 and 5 ($c\pi$ and $c\pi$++) to compute the Cπ-repair, and can see that the improved method largely outruns the naive method. Mainly, in situations of weakly conflicting ABoxes, which correspond to ABoxes with $(111, 387)$ conflicts for **u.5**, $(753, 3049)$ conflicts for **u1** and $(4505, 8327)$ for **u5**. For these ABoxes, the improved method reduces the running time to a mere 25% of the naive method's. This can be observed in Figure 1 [left], with $(753, 3049)$ conflicts. This also holds for large sized ABoxes, as ABox **u5** in Table 1. Moreover, for highly conflicting ABoxes, the improved method takes at most half of the time taken by the naive one. In these ABoxes, the π-repair becomes smaller and more assertions must be checked using the Cπ-repair characterization. We can observe in Figure 1 [left] that the π-repair is computed in a stable time. This is because it runs a constant time verification step for each ABox assertion.

Regarding the size of the obtained repairs, we observe that a large subset of assertions is discarded when the number of conflicts exceeds 10% of the size of the ABox (1268 for ABox **u.5**, 10388 for ABox **u1** and 57054 for ABox **u5**). This is confirmed by Figure 1 [right]. This outcome is expected. Moreover, it is true even for the IAR semantics, which is considered by nature to be more productive than possibilistic repairs (for more details on IAR, see [4, 1]).

Impact of the used partial order (Case 2)   In order to point out the main situations where the Cπ-repair is more productive than the π-repair, we illustrate the size of the π-repair (π), the closure of
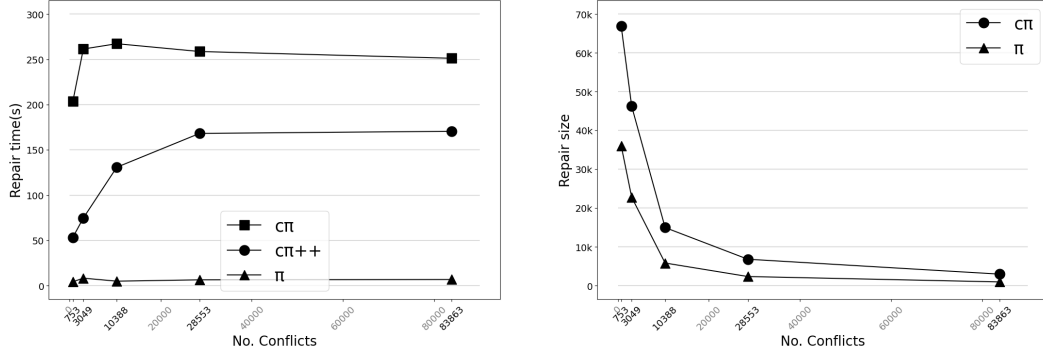
Figure 1: Evolution of repair time [left] and repair size [right] w.r.t. the number of conflicts (ABox **u1**).

the π-repair ($cl(\pi)$) and the additional assertions that are only in the Cπ-repair (only $c\pi$), as a function of the density of the used POS. Figure 2 [left] depicts the results of these executions. Note that in this case, we used ABox **u1** with 753, 3049 and 10388 conflicts. The evaluation of the density shows that lower probabilities, which indicate a higher presence of incomparabilities, lead to a more productive Cπ-repair. When the probability of having a strict preference between two degrees in the POS is lower than 0.5, the Cπ-repair is larger than the π-repair. Conversely, POSs with higher probabilities corresponds to a decreased likelihood of Cπ-repair being more productive than the π-repair, such partial orders approximate a total order. This result illustrates Lemma 2, with more incomparabilities, there is a higher chance for an assertion to have different supports dominating different conflicts. This is the case for the assertions that are only in the Cπ-repair.

We can also observe that, for POSs with many incomparabilities, the Cπ-repair returns more answers when the π-repair is empty. This can be observed in Figure 2 [left], with the densities 0.1 and 0.3. Furthermore, we concluded, trough multiple executions with different POS densities, that the Cπ-repair returns an important number of assertions in 80% of the cases where the π-repair contains less than 10% of the ABox assertions. In addition, for weakly conflicting ABoxes, the Cπ-repair provided additional assertions in 10% of the times. This percentage falls under 5% for the highly conflicting ABoxes.

Proportion of time spent by Algorithms 4 and 5 (Case 3) Figure 2 ([middle] and [right]) illustrates the running time taken by each step of Algorithms 4 and 5. For both algorithms, the computation of the ABox closure is achieved in constant time. The time taken to compute the set of conflicts increases slightly as their number increases. This also results in an increase in total time, as there are more conflicts to go through. In the improved method (Figure 2 [right]), computing the π-repair is run in constant time with respect to the changing number of conflicts, whereas computing its closure is more time-consuming in weakly conflicting ABoxes. Computing the supports involves processing each assertion in the ABox closure through multiple SQL queries to retrieve them. Although significantly faster in the improved algorithm, as shown in the results of Table 1, supports computation is the most time-intensive step of the process.

5. Concluding discussions

In this paper, we conducted an experimental study of data repair methods for partially ordered possibilistic knowledge bases, including three principal phases. Initially, it involved developing suitable algorithms for the computation of repairs. Subsequently, we delved into examining the inherent properties of the repair strategies and the algorithms' to enhance their efficiency. Finally, we tested the algorithms across various scenarios to evaluate their performance. We provided an algorithmic investigation, complemented by experimental analysis of the closure-based partially ordered possibilistic repair. Starting with the repair's characterization, we showed that leveraging certain properties allows for a more efficient repair process. Our experiments, conducted across
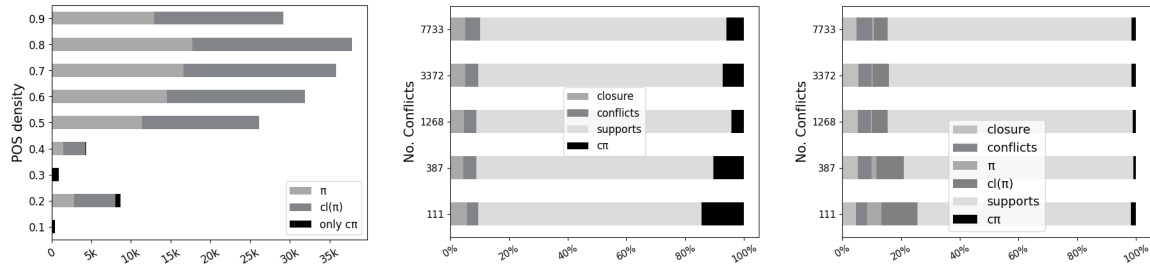
Figure 2: Proportion of assertions in each repair vs POS density [left]. Proportion of repair time for ABox **u.5**, using Algorithm 4 [middle] and Algorithm 5 [right].

diverse ABoxes, featuring a varying number of conflicts and configuration of partial orders, led to the insight that the Cπ-repair is particularly beneficial in cases where the partial order defined over the ABox incorporates numerous incomparabilities and its subset, the π-repair, includes fewer assertions. Furthermore, the algorithms we developed are capable of batch processing, enabling an incremental computation of repairs for larger ABoxes. Future work involves checking whether polynomial time complexity holds also in combined complexity.

References

[1] M. Bienvenu, C. Bourgaux, F. Goasdoué, Querying inconsistent description logic knowledge bases under preferred repair semantics, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 28, 2014, pp. 996–1002.

[2] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. F. Savo, Inconsistency-tolerant semantics for description logics, in: Web Reasoning and Rule Systemss, RR 2010, Bressanone/Brixen, Italy, 2010, pp. 103–117.

[3] M. Bienvenu, R. Rosati, Tractable approximations of consistent query answering for robust ontology-based data access, in: IJCAI, 2013, pp. 775–781.

[4] R. Rosati, M. Ruzzi, M. Graziosi, G. Masotti, Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies, in: The Semantic Web–ISWC 2012: 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II 11, Springer, 2012, pp. 337–349.

[5] S. Benferhat, Z. Bouraoui, K. Tabia, How to select one preferred assertional-based repair from inconsistent and prioritized DL-Lite knowledge bases?, in: International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina, 2015, pp. 1450–1456.

[6] S. Belabbes, S. Benferhat, J. Chomicki, Elect: An inconsistency handling approach for partially preordered lightweight ontologies, in: Logic Programming and Nonmonotonic Reasoning (LPNMR), Philadelphia, USA, 2019, pp. 210–223.

[7] S. Staworko, J. Chomicki, J. Marcinkowski, Prioritized repairing and consistent query answering in relational databases, AMAI 64 (2012) 209–246.

[8] M. Bienvenu, C. Bourgaux, Querying and repairing inconsistent prioritized knowledge bases: Complexity analysis and links with abstract argumentation, in: Principles of Knowledge Representation and Reasoning (KR), Virtual Event, 2020, pp. 141–151.

[9] A. Telli, S. Benferhat, M. Bourahla, Z. Bouraoui, K. Tabia, Polynomial algorithms for computing a single preferred assertional-based repair, KI-Künstliche Intelligenz 31 (2017) 15–30.

[10] S. Belabbes, S. Benferhat, Computing a possibility theory repair for partially preordered inconsistent ontologies, IEEE Transactions on Fuzzy Systems (2021) 1–10.

[11] A. Laouar, S. Belabbes, S. Benferhat, Tractable closure-based possibilistic repair for partially ordered dl-lite ontologies, in: European Conference on Logics in Artificial Intelligence, Springer, 2023, pp. 353–368.

[12] S. Belabbes, S. Benferhat, Characterizing the possibilistic repair for inconsistent partially ordered assertions, in: Information Processing and Management of Uncertainty in Knowledge-Based Systems - 19th International Conference, IPMU 2022, Milan, Italy, Proceedings, Part II, volume 1602, 2022, pp. 652–666.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The DL-Lite family, Journal of Automated Reasoning 39 (2007) 385–429.

[14] D. Calvanese, E. Kharlamov, W. Nutt, D. Zheleznyakov, Evolution of DL-Lite knowledge bases, in: International Semantic Web Conference (1), Shanghai, China, 2010, pp. 112–128.

[15] M. Bienvenu, C. Bourgaux, F. Goasdoué, Computing and explaining query answers over inconsistent dl-lite knowledge bases, Journal of Artificial Intelligence Research 64 (2019) 563–644.

[16] D. Krech, G. A. Grimnes, G. Higgins, J. Hees, I. Aucamp, N. Lindström, N. Arndt, A. Sommer, E. Chuc, I. Herman, A. Nelson, J. McCusker, T. Gillespie, T. Kluyver, F. Ludwig, P.-A. Champin, M. Watts, U. Holzer, E. Summers, W. Morriss, D. Winston, D. Perttula, F. Kovacevic, R. Chateauneu, H. Solbrig, B. Cogrel, V. Stuart, Rdflib, 2023. URL: https://zenodo.org/record/6845245.

[17] R. D. Hipp, SQLite, 2020. URL: https://www.sqlite.org/index.html.

[18] A. Chortaras, D. Trivela, G. Stamou, Optimized query rewriting for owl 2 ql, in: Automated Deduction–CADE-23: 23rd International Conference on Automated Deduction, Wrocław, Poland, July 31-August 5, 2011. Proceedings 23, Springer, 2011, pp. 192–206.

[19] G. Gutin, Acyclic digraphs, Classes of Directed Graphs (2018) 125–172.

[20] M. Scutari, Learning bayesian networks with the bnlearn r package, arXiv preprint arXiv:0908.3817 (2009).

[21] C. Lutz, I. Seylan, D. Toman, F. Wolter, The combined approach to OBDA: taming role hierarchies using filters, in: The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, Australia, 2013, pp. 314–330.