Comparing ABox Abduction Based on Minimal Hitting Set and MergeXplain

Katarína Fabianová, Júlia Pukancová and Martin Homola

Comenius University in Bratislava
Mlynská dolina, 84248 Bratislava
kfabianova11@gmail.com,{pukancova,homola}@fmph.uniba.sk

Abstract. We investigate an application of the MergeXplain algorithm on ABox abduction. This approach was recently proposed to address computational limitations of more traditional approaches, such as Reiter's Minimal Hitting Set algorithm. MergeXplain uses divide and conquer to search through the space of possible explanations more quickly, however this comes at a cost, as it is not complete. In this paper, we report on our preliminary experimental evaluation of both approaches in the context of finding explanations of ABox abduction problems in description logics. Finally, we also investigate how to combine both approaches in order to leverage on the improved effectivity of MergeXplain and still retain completeness.

Keywords: description logics · abduction · MergeXplain · minimal hitting set

1 Introduction

Abduction explains why some observation does not follow from a knowledge base (KB). This problem was originally introduced by Peirce [16], and it was studied also in the context of description logics (DL) [5,12,10,9,4,13,6,2,17,18,14,3]. We focus on ABox abduction [5] wherein both observations and explanations are extensional (i.e., they come in the form of ABox assertions).

Multiple existing works adopted Reiter's Minimal Hitting Set algorithm [19] exploiting a DL reasoner for satisfiability checking. This was proposed by Halland and Britz [10,9] and later extended by Pukancová and Homola [17,18] and Mrózek et al. [14] who have developed a black box approach and its proof-of-concept implementation.

The main advantage of Reiter's algorithm is its completeness. Also, thanks to properties of breath-first search it finds smaller explanations first [18], but to search through the complete space of explanations may simply take too long.

However, completeness is not necessarily required. In some applications it may be sufficient to find just one explanation. For such scenarios Junker [11] proposed QuickXplain (QXP), which uses *divide and conquer* to find one explanation more effectively. This method was further extended by Shchekotykhin et al. [20] into MergeXplain (MXP), capable of finding multiple explanations in one run.

In this work we report on our experiments with MXP-based abduction. We have implemented both methods (MHS and MXP) into an ABox abduction solver. Our implementation calls DL reasoners as a *black box* using OWL API similarly to Mrózek et

Table 1. Syntax and Semantics of $\mathcal{ALCHO}$

Concept	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}}$
complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \mid \exists y\,(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
Axiom	Syntax	Semantics
concept incl. (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role incl. (RIA)	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
neg. role assertion	$\neg R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}$

al. [14]. This allowed us to repeat all our experiments with Pellet [22], HermiT [21], and JFact [15].

We have conducted a preliminary experimental evaluation with two use cases. In the first, simpler use case we focused on explanations of size one. While MXP is incomplete in general, it is guaranteed to find all explanations of size one in a single run, hence such a comparison is interesting. In the second use case we did not limit the size of explanations. We simply set a timeout (of 12 hours) and compared the number of explanations found by each approach.

Our evaluation shows that in cases when one is only interested in explanations of size one MHS is more effective. In cases when larger explanations cannot be ruled out of consideration MXP was able to find over 80 % of explanations found by MHS before the timeout expired.

Finally, we have also investigated a combined approach, in which MXP is run repeatedly and MHS is used to steer this process in order to achieve completeness. We present the resulting algorithm. Empirical evaluation of this approach is subject to our ongoing work.

2 ABox Abduction in DL

For simplicity we will introduce $\mathcal{ALCHO}$ [1]. However any other DL may be used due to our *black box* approach. A vocabulary consists of countably infinite mutually disjoint sets of individuals $N_{\mathrm{I}} = \{a, b, \dots\}$, roles $N_{\mathrm{R}} = \{P, R, \dots\}$, and atomic concepts $N_{\mathrm{C}} = \{A, B, \dots\}$. Concepts are recursively built using constructors $\neg$, $\sqcap$, $\exists$, as shown in Table 1. A KB $\mathcal{K} = \mathcal{T} \cup \mathcal{A}$ consists of a a finite set of GCI and RIA axioms (in TBox $\mathcal{T}$), and a finite set of assertions (in ABox $\mathcal{A}$) as given in Table 1.

An interpretation is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}} \neq \emptyset$ is a domain, and the interpretation function $\cdot^{\mathcal{I}}$ maps each individual $a \in N_{\mathrm{I}}$ to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each atomic concept $A \in N_{\mathrm{C}}$ to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each role $R \in N_{\mathrm{R}}$ to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ in such a way that the interpretation of any complex concept is as given on the right-hand side of Table 1.

An interpretation $\mathcal{I}$ satisfies an axiom φ (denoted $\mathcal{I} \models \varphi$) if the respective constraint in Table 1 is satisfied. It is a model of a KB $\mathcal{K}$ (denoted $\mathcal{I} \models \mathcal{K}$) if $\mathcal{I} \models \varphi$ for all $\varphi \in \mathcal{K}$. A KB $\mathcal{K}$ is consistent, if there is at least one interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$. $\mathcal{K}$ entails an axiom φ (denoted $\mathcal{K} \models \varphi$) if $\mathcal{I} \models \varphi$ for each $\mathcal{I} \models \mathcal{K}$.

Finally, we define $\neg\varphi := \neg C(a)$ if $\varphi = C(a)$; $\neg\varphi := C(a)$ if $\varphi = \neg C(a)$; $\neg\varphi := \neg R(a,b)$ if $\varphi = R(a,b)$; $\neg\varphi := R(a,b)$ if $\varphi = \neg R(a,b)$.

In *ABox abduction* [5], we are given a KB $\mathcal{K}$ and an observation O consisting of an ABox assertion. The task is to find an explanation $\mathcal{E}$, again, consisting of ABox assertions, such that $\mathcal{K} \cup \mathcal{E} \models O$. However, the set of all ABox expressions may be too broad to draw the explanations from (after all, it is infinite), hence we typically consider some set of *abducibles* Abd. In this work we will limit the explanations to atomic and negated atomic concept assertions; so we set $\text{Abd} := \{A(a), \neg A(a) \mid A \in N_{\text{C}}, a \in N_{\text{I}}\}$. Note that we do not limit the observations in any way, apart from allowing only one (possibly complex) ABox assertion.

Definition 1 (ABox Abduction Problem). *Let* Abd *be a set of ABox assertions. An ABox abduction problem is a pair $\mathcal{P} = (\mathcal{K}, O)$ such that $\mathcal{K}$ is a knowledge base in DL and O is an ABox assertion. A solution of $\mathcal{P}$ (also called explanation) is any finite set $\mathcal{E} \subseteq$* Abd *of ABox assertions such that $\mathcal{K} \cup \mathcal{E} \models O$.*

While Definition 1 establishes the basic reasoning mechanism of abduction, some of the explanations it permits may be unintuitive. According to Elsenbroich et al. [5] given $\mathcal{P} = (\mathcal{K}, O)$ and its solution $\mathcal{E}$:

1. $\mathcal{E}$ is *consistent* if $\mathcal{K} \cup \mathcal{E}$ is a consistent KB,
2. $\mathcal{E}$ is *relevant* if $\mathcal{E} \not\models O$,
3. $\mathcal{E}$ is *explanatory* if $\mathcal{K} \not\models O$.

Indeed an explanation should be consistent, as anything follows from inconsistency; and so, an explanation that makes $\mathcal{K}$ inconsistent does not really explain the observation. It should be relevant – it should not imply the observation directly without requiring the KB $\mathcal{K}$ at all. And it should be explanatory, that is, we should not be able to explain the observation without it.

Only explanations which satisfy all three of the above conditions will be called *desired*. In addition, in order to avoid excess hypothesizing, minimality is required.

Definition 2 (Minimality). *Assume an ABox abduction problem $\mathcal{P} = (\mathcal{K}, O)$. Given two solutions $\mathcal{E}$ and $\mathcal{E}'$ of $\mathcal{P}$, we say that $\mathcal{E}$ is (syntactically)* smaller *than $\mathcal{E}'$ if $\mathcal{E} \subseteq \mathcal{E}'$.*[1] *We further say that a solution $\mathcal{E}$ of $\mathcal{P}$ is* syntactically minimal *if there is no other solution $\mathcal{E}'$ of $\mathcal{P}$ that is smaller than $\mathcal{E}$.*

3 Our Approach

We first describe the MHS algorithm and then the MergeXplain algorithm. Both were implemented and evaluated as reported in Section 4. Finally we also describe how both algorithms may be combined to obtain a more effective yet still complete approach.

[1] Note that before we compare two solutions $\mathcal{E}$ and $\mathcal{E}'$ of $\mathcal{P}$ syntactically, we typically normalize the assertions w.r.t. (outermost) concept conjunction: as $C \sqcap D(a)$ is equivalent to the pair of assertions $C(a)$ and $D(a)$, we replace the former form by the latter while possible.

3.1 Minimal Hitting Set Algorithm

This approach is based on the fact that $\mathcal{E}$ is an explanation of $\mathcal{P} = (\mathcal{K}, O)$ if and only if $\mathcal{K} \cup \mathcal{E} \models O$, i.e., if and only if $\mathcal{K} \cup \mathcal{E} \cup \{\neg O\}$ is inconsistent. Assume $\mathcal{K} \cup \{\neg O\}$ is consistent and the set of all its models is $\mathcal{M}$. To find some explanation, we may simply collect in $\mathcal{E}$ assertions that invalidate each model $M \in \mathcal{M}$. However we want to draw these assertions from the set of abducibles Abd. Hence it suffices to find a hitting set (MHS) of the set $\{\mathrm{Abd}(M) \mid M \in \mathcal{M}\}$ where $\mathrm{Abd}(M) = \{\phi \in \mathrm{Abd} \mid M \not\models \phi\}$ is a set of expressions that invalidate M and are part of Abd.

In fact, as proved by Reiter [19], the set of all minimal explanations corresponds to the set of all minimal hitting sets of $\{\mathrm{Abd}(M) \mid M \in \mathcal{M}\}$, modulo negation.

The algorithm works by constructing a HS-tree $T = (V, E, L)$, a labelled tree in which each node is labelled by $\mathrm{Abd}(M)$ for some model M of $\mathcal{K} \cup \{\neg O\}$ and whose edges are labelled by elements of the parent node's label. The HS-tree has the property that the node label $L(n)$ and the union of the edge-labels $H(n)$ on the path from root r to each node n are disjoint. If n cannot be extended by any successor satisfying this property then $H(n)$ corresponds to a hitting set.

In addition, *pruning* is applied during the HS-tree construction in order to eliminate non-minimal hitting sets. The most obvious case is when a node n already corresponds to a hitting set $H(n)$ and there is another node n' with $H(n) \subseteq H(n')$. Any such n' can be pruned. A pruned HS-tree (i.e., one on which all pruning conditions were applied), once completed, contains all minimal hitting sets [19]. In addition we also prune nodes which correspond to undesired explanations [17].

The resulting algorithm is given in Algorithm 1. This algorithm is sound and complete [19,17,18].

Theorem 1. *The MHS algorithm is sound and complete (i.e., it returns the set $S_{\mathcal{E}}$ of all minimal desired explanations of $\mathcal{K}$ and O.)*

The fact that MHS explores the search space using breadth-first search (BFS) allows to limit the search for explanations by maximum size. The algorithm is still complete w.r.t. any given target size [18].

3.2 MergeXplain Algorithm

MXP is listed in Algorithm 2. Both QXP and MXP were originally designed to find minimal inconsistent subsets (dubbed *conflicts*) of an overconstrained knowledge base $\mathcal{K} = \mathcal{B} \cup \mathcal{C}$, where $\mathcal{K}$ is the consistent background theory and $\mathcal{C}$ is the "suspicious" part from which the conflicts are drawn. This can be immediately adopted for ABox abduction: in order to find explanations for an abduction problem $\mathcal{P} = (\mathcal{K}, O)$ one needs to call $\mathrm{MXP}(\mathcal{K} \cup \{\neg O\}, \mathrm{Abd})$. This observation allows us to adopt the following result from Shchekotykhin et al. [20]:

Theorem 2. *Assume an ABox abduction problem $\mathcal{P} = (\mathcal{K}, O)$ and a set of abducibles* Abd. *If there is at least one explanation $\gamma \subseteq$* Abd *of $\mathcal{P}$ then calling* $\mathrm{MXP}(\mathcal{K} \cup \{\neg O\}, \mathrm{Abd})$ *returns a nonempty set Γ of explanations of $\mathcal{P}$.*

Algorithm 1 MHS($\mathcal{K}$,O)

Require: knowledge base $\mathcal{K}$, observation O
Ensure: set $\mathcal{S}_\mathcal{E}$ of all explanations of $\mathcal{P} = (\mathcal{K}, O)$ of the class Abd

$M \leftarrow$ a model M of $\mathcal{K} \cup \{\neg O\}$
if $M =$ `null` **then**
    return `"nothing to explain"`
end if
create new HS-tree $T = (V, E, L)$ with root r
label r by $L(r) \leftarrow \text{Abd}(M)$
for each $\sigma \in L(r)$ create a successor n_σ of r and label the resp. edge by σ
$\mathcal{S}_\mathcal{E} \leftarrow \{\}$
while there is next node n in T w.r.t. BFS **do**
    if n can be pruned **then**
        prune n
    else if there is a model M of $\mathcal{K} \cup \{\neg O\} \cup H(n)$ **then**
        label n by $L(n) \leftarrow \text{Abd}(M)$
    else
        $\mathcal{S}_\mathcal{E} \leftarrow \mathcal{S}_\mathcal{E} \cup \{H(n)\}$
    end if
    for each $\sigma \in L(n)$ create a successor n_σ of n and label the resp. edge by σ
end while
return $\mathcal{S}_\mathcal{E}$

Thus MXP is sound, and it always finds at least one explanation if at least one exists. However it is not complete, especially in cases of abduction problems which have multiple partially overlapping explanations.

Example 1. Let $\mathcal{K} = \{A \sqcap B \sqsubseteq D, A \sqcap C \sqsubseteq D\}$ and let $O = D(a)$. Let us ignore negated ABox expressions and start with Abd $= \{A(a), B(a), C(a)\}$. There are two minimal explanations of $\mathcal{P} = (\mathcal{K}, O)$: $\{A(a), B(a)\}$, and $\{A(a), C(a)\}$. Calling MXP($\mathcal{K} \cup \{\neg O\}$, Abd), it passes the initial tests and calls FINDCONFLICTS($\mathcal{K} \cup \{\neg O\}$, Abd).

FINDCONFLICTS needs to decide how to split $C =$ Abd into C_1 and C_2. Let us assume the split was $C_1 = \{A(a)\}$ and $C_2 = \{B(a), C(a)\}$. Since both C_1 and C_2 are now conflict-free w.r.t. $\mathcal{K} \cup \{\neg O\}$, the two consecutive recursive calls return $\langle C_1', \emptyset\rangle$ and $\langle C_2', \emptyset\rangle$ where $C_1' = \{A(a)\}$ and $C_2' = \{B(a), C(a)\}$.

In the while loop, GETCONFLICT($\mathcal{K} \cup \{\neg O\} \cup \{B(a), C(a)\}$, $\{B(a), C(a)\}$, $\{A(a)\}$) returns $X = \{A(a)\}$ while GETCONFLICT($\mathcal{K} \cup \{\neg O\} \cup \{A(a)\}$, $\{A(a)\}$, $\{B(a), C(a)\}$) returns $B(a)$, and hence the first confclit $\gamma = \{A(b), B(a)\}$ is found and added into Γ.

However, consecutively $A(a)$ is removed from C_1' leaving it empty, and thus the other conflict is not found and $\Gamma = \{\{A(b), B(a)\}\}$ is returned.

3.3 Combined Approach

As showed by the example, MergeXplain is not always complete. However, both approaches can be combined in order to regain completeness. The idea is to call MXP repeatedly and use the construction of the HS-tree to steer this process and to guarantee completeness.

Algorithm 2 MXP($\mathcal{B}$,$\mathcal{C}$)

Input: $\mathcal{B}$: Background theory, $\mathcal{C}$: the set of possibly faulty constraints
Output: Γ, a set of minimal conflicts

1: **if** $\neg isConsistent(\mathcal{B})$ **then**
2:     **return** `"no solution"`
3: **else if** $isConsistent(\mathcal{B} \cup \mathcal{C})$ **then**
4:     **return** $\emptyset$
5: **end if**
6: $\langle \_, \Gamma \rangle \leftarrow$ FINDCONFLICTS($\mathcal{B}, \mathcal{C}$)
7: **return** Γ

8: **function** FINDCONFLICTS($\mathcal{B}, \mathcal{C}$) **returns** a tuple $\langle \mathcal{C}', \Gamma \rangle$
9:     **if** $isConsistent(\mathcal{B} \cup \mathcal{C})$ **then**
10:         **return** $\langle \mathcal{C}, \emptyset \rangle$
11:     **else if** $|\mathcal{C}| = 1$ **then**
12:         **return** $\langle \emptyset, \{\mathcal{C}\} \rangle$
13:     **end if**
14:     Split $\mathcal{C}$ into disjoint, non-empty sets $\mathcal{C}_1$ and $\mathcal{C}_2$
15:     $\langle \mathcal{C}'_1, \Gamma_1 \rangle \leftarrow$ FINDCONFLICTS($\mathcal{B}, \mathcal{C}_1$)
16:     $\langle \mathcal{C}'_2, \Gamma_2 \rangle \leftarrow$ FINDCONFLICTS($\mathcal{B}, \mathcal{C}_2$)
17:     $\Gamma \leftarrow \Gamma_1 \cup \Gamma_2$
18:     **while** $\neg isConsistent(\mathcal{C}'_1 \cup \mathcal{C}'_2 \cup \mathcal{B})$ **do**
19:         $X \leftarrow$ GETCONFLICT($\mathcal{B} \cup \mathcal{C}'_2, \mathcal{C}'_2, \mathcal{C}'_1$)
20:         $\gamma \leftarrow X \cup$ GETCONFLICT($\mathcal{B} \cup X, X, \mathcal{C}'_2$)
21:         $\mathcal{C}'_1 \leftarrow \mathcal{C}'_1 \setminus \{a\}$ where $a \in X$
22:         $\Gamma \leftarrow \Gamma \cup \{\gamma\}$
23:     **end while**
24:     **return** $\langle \mathcal{C}'_1 \cup \mathcal{C}'_2, \Gamma \rangle$
25: **end function**

26: **function** GETCONFLICT($\mathcal{B}, D, \mathcal{C}$)
27:     **if** $D \neq \emptyset \wedge \neg isConsistent(\mathcal{B})$ **then**
28:         **return** $\emptyset$
29:     **else if** $|\mathcal{C}| = 1$ **then**
30:         **return** $\mathcal{C}$
31:     **end if**
32:     Split $\mathcal{C}$ into disjoint, non-empty sets $\mathcal{C}_1$ and $\mathcal{C}_2$
33:     $D_2 \leftarrow$ GETCONFLICT($\mathcal{B} \cup \mathcal{C}_1, \mathcal{C}_1, \mathcal{C}_2$)
34:     $D_1 \leftarrow$ GETCONFLICT($\mathcal{B} \cup D_2, D_2, \mathcal{C}_1$)
35:     **return** $D_1 \cup D_2$
36: **end function**

The combined algorithm MHS-MXP, presented in Algorithm 3, amends the function FINDCONFLICTS with additional third parameter $\delta \subseteq \mathcal{C}$ that guarantees that if δ is a conflict for $\mathcal{B}$ (i.e., if $\mathcal{B} \cup \delta$ is inconsistent) then $\delta \in \Gamma$, where Γ is the set of conflicts returned by FINDCONFLICTS($\mathcal{B}, \mathcal{C}, \delta$). This is assured by the modifications in lines 27 and 30. These changes are satisfactory to guarantee that if δ is a conflict for $\mathcal{B}$, it is not lost from the output. The GETCONFLICT function did not require any modifications.

Algorithm 3 MHS-MXP($\mathcal{K}$,O)

Require: knowledge base $\mathcal{K}$, observation O
Ensure: set $\mathcal{S}_{\mathcal{E}}$ of all explanations of $\mathcal{P} = (\mathcal{K}, O)$ of the class Abd

1: **if** $\neg isConsistent(\mathcal{K} \cup \{\neg O\})$ **then**
2:     **return** `"nothing to explain"`
3: **end if**
4: create new HS-tree $T = (V, E, L)$ with root r
5: $\mathcal{C} \leftarrow \{\}$
6: **while** there is next node n in T w.r.t. BFS s.t. $L(n) \subseteq$ Abd, $L(n) \neq \emptyset$ **do**
7:     **if** $\gamma \in \mathcal{C}$ s.t. $\gamma \subseteq H(n)$ **then**
8:         $n \leftarrow \emptyset$ ▷ n is pruned
9:     **else**
10:         $\langle \_, \Gamma \rangle \leftarrow$ FINDCONFLICTS($\mathcal{K} \cup \{\neg O\}$, Abd, $H(n)$)
11:         **if** $\Gamma = \emptyset$ go to 23 **end if**
12:         **for all** $\gamma \in \Gamma$ **do**
13:             **for all** ordered sequences $\langle a_1, \dots, a_n \rangle$ s.t. $\{a_1, \dots, a_n\} = \gamma$ **do**
14:                 ADDPATH(r,$\langle a_1, \dots, a_n \rangle$)
15:             **end for**
16:             **for all** $a \in L(n)$ s.t. there is no n' with $L((n, n')) = a$ **do**
17:                 add node n', $L((n, n')) \leftarrow a$
18:             **end for**
19:             $\mathcal{C} \leftarrow \mathcal{C} \cup \Gamma$
20:         **end for**
21:     **end if**
22: **end while**
23: **return** $\mathcal{S}_{\mathcal{E}} \leftarrow \{\gamma \in \mathcal{C} \mid \gamma \text{ is desired}\}$
24: **function** FINDCONFLICTS($\mathcal{B}, \mathcal{C}, \delta$) **returns** a tuple $\langle \mathcal{C}', \Gamma \rangle$
25:     **if** $isConsistent(\mathcal{B} \cup \mathcal{C})$ **then**
26:         **return** $\langle \mathcal{C}, \emptyset \rangle$
27:     **else if** $|\mathcal{C}| = 1$ or $\mathcal{C} = \delta$ **then**
28:         **return** $\langle \emptyset, \{\mathcal{C}\} \rangle$
29:     **end if**
30:     Split $\mathcal{C}$ into disjoint, non-empty sets $\mathcal{C}_1$ and $\mathcal{C}_2$ s.t. if $\delta \subseteq \mathcal{C}$ then $\delta \subseteq \mathcal{C}_2$
31:     $\langle \mathcal{C}'_1, \Gamma_1 \rangle \leftarrow$ FINDCONFLICTS($\mathcal{B}, \mathcal{C}_1, \delta$)
32:     $\langle \mathcal{C}'_2, \Gamma_2 \rangle \leftarrow$ FINDCONFLICTS($\mathcal{B}, \mathcal{C}_2, \delta$)
33:     $\Gamma \leftarrow \Gamma_1 \cup \Gamma_2$
34:     **while** $\neg isConsistent(\mathcal{C}'_1 \cup \mathcal{C}'_2 \cup \mathcal{B})$ **do**
35:         $X \leftarrow$ GETCONFLICT($\mathcal{B} \cup \mathcal{C}'_2, \mathcal{C}'_2, \mathcal{C}'_1$)
36:         $\gamma \leftarrow X \cup$ GETCONFLICT($\mathcal{B} \cup X, X, \mathcal{C}'_2$)
37:         $\mathcal{C}'_1 \leftarrow \mathcal{C}'_1 \setminus \{a\}$ where $a \in X$
38:         $\Gamma \leftarrow \Gamma \cup \{\gamma\}$
39:     **end while**
40:     **return** $\langle \mathcal{C}'_1 \cup \mathcal{C}'_2, \Gamma \rangle$
41: **end function**
42: **function** ADDPATH(n, $\langle a_1, \dots, a_n \rangle$)
43:     **if** there is no n' in T s.t. $L(n, n') = a_1$ **then**
44:         add node n', $L((n, n')) \leftarrow a$, $L(n') \leftarrow L(n)$
45:     **end if**
46:     **if** $n > 1$ **then**
47:         ADDPATH(n',$\langle a_2, \dots, a_n \rangle$)
48:     **else**
49:         $L(n') \leftarrow \emptyset$
50:     **end if**
51: **end function**

MHS-MXP starts by checking if $\mathcal{K} \models O$ (i.e., if $\mathcal{K} \cup \{\neg O\}$ is inconsistent) in which case there is nothing to explain. It then constructs the HS-tree with the following modifications: (a) Nodes are not labelled by subsets of Abd, instead the successor edges are drawn from the whole set Abd. Only nodes corresponding to found conflicts and to pruned nodes are labelled by $\emptyset$ to cut further search. (b) Nodes are explored by BFS, for each node n we call FINDCONFLICTS($\mathcal{K} \cup \{\neg O\}$, Abd, $H(n)$) where by passing $H(n)$ as parameter we assure that, if it is a conflict, it is not omitted from the resulting set of conflicts Γ. (c) If FINDCONFLICTS did not find any conflicts (i.e., if $\Gamma = \emptyset$) we terminate the search. We can do this because MXP always return some conflict, if one exists [20]. Hence we can be sure that the search is over. (d) The HS-tree is enriched by all paths found in Γ which are now verified minimal conflicts and hence (the corresponding leaf-nodes) are omitted from future exploration. This is assured by labelling the ends of these paths by $\emptyset$. (e) From now on all paths corresponding to their supersets will immediately be pruned when encountered by the BFS search – they correspond to nonminimal conflicts. (f) Finally we filter out conflicts corresponding to undesired explanations.

Theorem 3. *The MHS-MXP algorithm is sound and complete (i.e., it returns the set $\mathcal{S}_{\mathcal{E}}$ of all minimal desired explanations of $\mathcal{K}$ and O.)*

The soundness follows from the soundness of MXP. The completeness follows from the observation that for every $\gamma \subseteq$ Abd that is an explanation of $\mathcal{P} = (\mathcal{K}, O)$, the HS-tree contains a path respective to some leaf node n s.t. $H(n) = \gamma$ as γ was only subtracted from the node labels once at least one such node was added to the HS-tree.

Note that one additional optimization is possible: after FINDCONFLICTS is called for the first time and returns Γ, it is safe to reduce Abd by removing all $\gamma \in \Gamma$ such that $|\gamma| = 1$. This is because all minimal conflicts involving these abducibles were now found. This reduces the search space, and in the special case when all explanations are of size one the algorithm will terminate after two calls of FINDCONFLICTS.

4 Evaluation

A preliminary experimental evaluation was conducted with implementations of MHS and MXP, both paired with three DL reasoners – Pellet, HermiT, and JFact. Both algorithms are implemented in Java and communicate with the reasoners through OWL API. The source code of both implementations is available online.[2]

The evaluation is split into two experiments. Experiment 1 is focused on computing explanations of size one. In this case MHS can be made more effective by bounding the HS-tree depth, and on the other hand MXP is complete in this case. Experiment 2 was conducted without any constraints on the size of explanations, but a timeout needed to be set. Both experiments were focused on comparing execution times between the two approaches and the three reasoners. Each time was computed as an average value from ten runs with ten different observations.

[2] `https://github.com/katuskaa/MasterThesis`

4.1 Dataset and Methodology

Three ontologies were chosen. The Family ontology[3], is our own ontology of family relations. It is smaller, but it is particularly useful in this use case as it generates a number of explanations of size higher than one. LUBM ontology [8], is a standard benchmark. Beer ontology[4] was chosen. Both LUBM and Beer were chosen because of their larger size compared to the Family ontology, but on the other hand as in the case of many real world ontologies their axiomatic structure is less complex which implies that most if not all explanations are of size one.

Table 2. Parameters of the ontologies

Ontology	Concepts	Roles	Individuals	Axioms
Family ontology	10	1	0	28
Beer ontology	58	9	0	165
LUBM	43	25	0	243

All experiments were done on a virtual machine with 8 cores (16 threads) of the processor Intel®Xeon®CPU E5-2695 v4, 32 GB RAM, 2.10GHz, running Ubuntu 18.04.1, while the maximum Java heap size was set to 4GB. Execution times were measured in Java using `ThreadMXBean` from the `java.lang.management` package. We used *user* time, that is the actual time without system overhead.

4.2 Experiment 1

The first experiment is focused on comparing both methods (MHS and MXP) on a simplified task of finding all explanations of size one. We focus on this class of explanations for two reasons. Firstly, many real-world ontologies feature only weak axiomatization, if any, implying that most explanations, if not all, will be of size one.

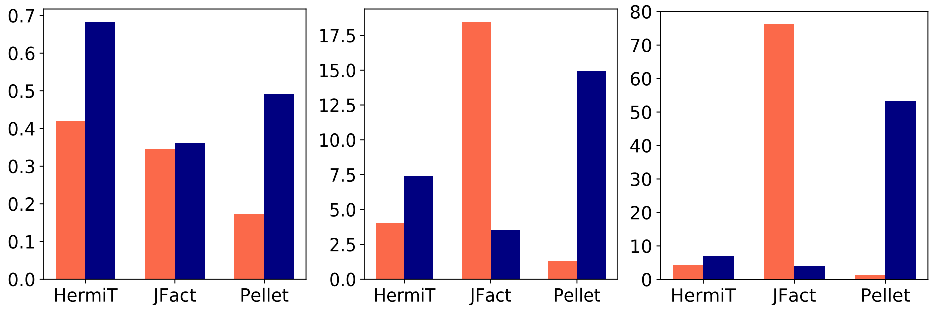
Secondly, both MHS and MXP can be used to find all explanations of size one quite effectively. MHS – which is complete but normally quite ineffective – can be depth-bound and stopped after it explored the search space up to size one, which takes considerably less time than to explore the rest of the search space. MXP, in turn, is incomplete in general, but it does not lose any explanations of size one.

For each ontology Figure 1 plots the average time of execution over ten observations for each of the three reasoners and for both methods. In case of the smallest Family ontology Figure 1 (a) shows that MHS is always quicker than MXP regardless of the DL reasoner used. Among the reasoners Pellet is the quickest when paired with MHS, however JFact is the quickest when paired with MXP. In all three cases MXP has found one additional explanation (of size 2) than MHS for one of the observations.

The results for medium-sized Beer ontology are plotted in Figure 1 (b). We observe that MHS is quicker than MXP with Pellet and HermiT, but in case of JFact MXP is quicker. Again, Pellet is the quickest when paired with MHS, and JFact is the quickest when paired with MXP, however the performance of HermiT is improved for this

[3] `http://dai.fmph.uniba.sk/~pukancova/aaa/ont/family2.owl`

[4] `https://www.cs.umd.edu/projects/plus/SHOE/onts/beer1.0.html`



(a) Family (b) Beer (c) LUBM

Fig. 1. Result times in seconds for Exp. 1: MHS MXP

larger ontology. Both approaches have found the same amount of explanations for each observation.

The LUBM ontology is the largest one in our use case. The plot in Figure 1 (c) shows that the results are essentially similar to the case of the Beer ontology. The only two cases which took significantly greater time were those of MHS/JFact and MXP/Pellet. We can also observe that the comparative performance of HermiT is further improved. Again, both approaches have found the same amount of explanations for each observation.

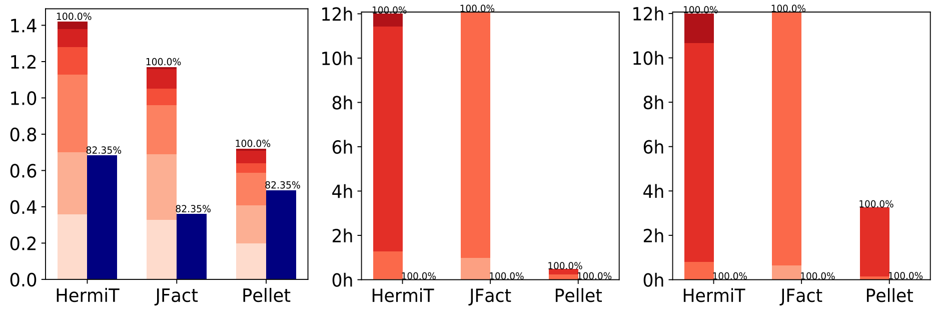
4.3 Experiment 2

The goal of this experiment was to compute as much explanations as possible using MHS and MXP. We compare execution times and the number of explanations found. As runtimes of MHS (especially for larger ontologies) may be too high, a timeout of 12 hours (43200 seconds) was set. MXP terminated way faste in all runs, however, as we know, it does not explore the whole search space. The results are plotted in Figure 2.

The Family ontology (a) is much smaller compared to the other two, hence all runs finished before the timeout. Its richer axiomatization generates a number of explanations of size higher than one. Hence while MXP took shorter time than MHS it found a smaller number of explanations. Still, on average it found 82.35 % of the explanations found by MHS. On this smaller ontology once again Pellet was the quickest when paired with MHS while JFact was the quickest when paired with MXP.

The results for the Beer and LUBM ontologies (Figures 2, b–c) are similar. They show that even in case of medium size ontologies it simply is not feasible for MHS to explore the whole search space as it either reached the timeout (the cases of HermiT and JFact) or it exceeded the memory (the case of Pellet).[5] Note that it may be feasible to use MHS in cases when the set of abducibles is further reduced, i.e., if the user knows or is able to guess beforehand in which part of the search space to look for possible explanations. Such cases are out of the scope of this paper.

[5] Note that if the time out was reached we recorded 12 hours, but if the memory was exceeded we recorded the time at the end of the *last completed depth* of the HS-tree – hence the results for Pellet in Figure 2 (b–c) are only seemingly good.



(a) Family (b) Beer (c) LUBM

Fig. 2. Results in seconds (a) and hours (b–c) for Exp. 2: MHS (depth 1–6) MXP

In contrast, MXP only took a fraction of this time. (Note that while exact times of MXP runs are not readable from Figure 2 (b–c) due to scale, they are the same as in Figure 1 (b–c)). Both approaches found exactly all explanations of size one in each run (and none other). Since both ontologies feature only weak axiomatization it is unlikely that there are any larger explanations.

Looking at the MHS results of the three reasoners we observe that HermiT was the most successful in terms of being able to explore the largest part of the search space before the timeout. Interestingly, this is in contrast with the results on the Family ontology which may indicate that it likely features some initial overhead which is outweighed when the reasoning task grows larger.

5 Conclusions

In this work we have focused on comparison of MHS and MXP on the task of ABox abduction. We have implemented both approaches and conducted a preliminary evaluation. Our evaluation shows that in cases when there are only explanations of size one, or one is only interested in this size of explanations MHS is more effective. In cases when larger explanations cannot be ruled out MXP can be used for fast but incomplete query; in our tests we were able to obtain over 80 % of explanations in this way. MHS is complete but it is much slower. Even on medium size ontologies such as Beer and LUBM MHS did not terminate before the 12 hour timeout.

As we paired our solvers with three different DL reasoners in all experiments, we were able to compare the performance of these reasoners as well. Out of them HermiT was the most effective as it was able to search through the largest part of the search space before the 12 hour timeout expired.

We have also described a combined approach which leverages on the effectivity of MXP and uses MHS to steer the search and retain completeness. Empirical evaluation of this approach is an ongoing effort.

Acknowledgements. This work presents the results of the Master's thesis by Katarína Fabianová [7]. It was supported by the Slovak national project VEGA 1/0778/18.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Castano, S., Espinosa Peraldí, I.S., Ferrara, A., Karkaletsis, V., Kaya, A., Möller, R., Montanelli, S., Petasis, G., Wessel, M.: Multimedia interpretation for dynamic ontology evolution. J. Log. Comput. 19(5), 859–897 (2009)
3. Del-Pinto, W., Schmidt, R.A.: Forgetting-based abduction in $\mathcal{ALC}$. In: Proceedings of the Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2017), Dresden, Germany. CEUR-WS, vol. 2013, pp. 27–35 (2017)
4. Du, J., Qi, G., Shen, Y., Pan, J.Z.: Towards practical ABox abduction in large description logic ontologies. Int. J. Semantic Web Inf. Syst. 8(2), 1–33 (2012)
5. Elsenbroich, C., Kutz, O., Sattler, U.: A case for abductive reasoning over ontologies. In: Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, GA, US. CEUR-WS, vol. 216 (2006)
6. Espinosa Peraldí, I.S., Kaya, A., Möller, R.: Formalizing multimedia interpretation based on abduction over description logic ABoxes. In: Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK. CEUR-WS, vol. 477 (2009)
7. Fabianová, K.: Optimization of an abductive reasoner for description logics. Master's thesis, Comenius University in Bratislava (2019), to appear
8. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2-3), 158–182 (2005)
9. Halland, K., Britz, K.: Abox abduction in $\mathcal{ALC}$ using a DL tableau. In: 2012 South African Institute of Computer Scientists and Information Technologists Conference, SAICSIT '12, Pretoria, South Africa. pp. 51–58 (2012)
10. Halland, K., Britz, K.: Naïve ABox abduction in $\mathcal{ALC}$ using a DL tableau. In: Proceedings of the 2012 International Workshop on Description Logics, DL 2012, Rome, Italy. CEUR-WS, vol. 846 (2012)
11. Junker, U.: QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, San Jose, California, US. pp. 167–172. AAAI Press (2004)
12. Klarman, S., Endriss, U., Schlobach, S.: ABox abduction in the description logic $\mathcal{ALC}$. Journal of Automated Reasoning 46(1), 43–80 (2011)
13. Ma, Y., Gu, T., Xu, B., Chang, L.: An ABox abduction algorithm for the description logic $\mathcal{ALCI}$. In: Intelligent Information Processing VI – 7th IFIP TC 12 International Conference, IIP 2012, Guilin, China. Proceedings. IFIP AICT, vol. 385, pp. 125–130. Springer (2012)
14. Mrózek, D., Pukancová, J., Homola, M.: ABox abduction solver exploiting multiple DL reasoners. In: Proceedings of the 31st International Workshop on Description Logics, Tempe, Arizona, US. CEUR-WS, vol. 2211 (2018)
15. Palmisano, I.: JFact DL reasoner. `http://jfact.sourceforge.net/`
16. Peirce, C.S.: Deduction, induction, and hypothesis. Popular science monthly 13, 470–482 (1878)
17. Pukancová, J., Homola, M.: Tableau-based ABox abduction for the $\mathcal{ALCHO}$ description logic. In: Proceedings of the 30th International Workshop on Description Logics, Montpellier, France. CEUR-WS, vol. 1879 (2017)
18. Pukancová, J., Homola, M.: ABox abduction for description logics: The case of multiple observations. In: Proceedings of the 31st International Workshop on Description Logics, Tempe, Arizona, US. CEUR-WS, vol. 2211 (2018)

19. Reiter, R.: A theory of diagnosis from first principles. Artificial intelligence 32(1), 57–95 (1987)
20. Shchekotykhin, K.M., Jannach, D., Schmitz, T.: MergeXplain: Fast computation of multiple conflicts for diagnosis. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina. AAAI Press (2015)
21. Shearer, R., Motik, B., Horrocks, I.: Hermit: A highly-efficient OWL reasoner. In: Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, Karlsruhe, Germany. CEUR-WS, vol. 432 (2008)
22. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)