Ontology Repair Through Partial Meet Contraction

Raphael Cóbe, Renata Wassermann
University of São Paulo
São Paulo, Brazil
{rmcobe,renata}@ime.usp.br

Abstract

The process of building an ontology, be it from scratch or through reuse and combination of other ontologies, is known to be susceptible to modeling errors. Ontology debugging and repair techniques have attracted attention in the last decade due to the popularization of the use of ontologies written in OWL. Belief Change deals with the problem of removing or adding new information to a knowledge base in a consistent way. In this paper, we look at the belief change operation known as *partial meet contraction* as a construction for ontology repair. We propose heuristics to improve the performance of such operation and compare them to an existing implementation and approaches based on finding minimal justifications or explanations by means of experiments with automatically generated ontologies and real world ontologies from the BioPortal.

1 Introduction

The ontology building task is a non trivial, error prone task which has, mostly, to be carried out by experts on the domain being modeled. Most ontology editors offer a connection to a reasoner in order to test for conflicts. The nature of such conflicts can be a logical one, where the resulting ontology is considered inconsistent or incoherent (i.e. the interpretation of some concept is necessarily empty), or even an unwanted entailment that should not be valid.

The problem we address in this paper is that of going one step further and actually providing the ontology developer solutions for the unwanted behavior, guaranteeing that the resulting ontology is *safe*, i.e., free of conflicts.

In this paper, we propose the use of belief base contraction to debug and repair ontologies. Belief change [Alchourron *et al.*, 1985; Hansson, 1999] is the area of knowledge representation that deals with adding or removing information from a knowledge base. These changes are often non-trivial, as one wants to preserve logical consistency. There are typically two constructions used in the belief change literature: one based on finding minimal conflict sets and deleting at least one formula from each, and one based on finding maximal conflict-free sets. The first construction is called *kernel contraction* and the second one is known as *partial meet contraction*.

There have been several proposals to apply belief change operators to description logics [Flouris, 2006; Ribeiro, 2013] as well as implementations for ontology repair [Haase *et al.*, 2005; Kalyanpur, 2006; Schlobach *et al.*, 2007; Qi *et al.*, 2008; Horridge, 2011]. Virtually all implemented solutions are based on the idea of kernel construction, i.e., finding minimal conflicting sets. However, in order to solve a conflict, one needs to compute all kernels and then remove at least one element of each, while in the partial meet approach, if there are not enough resources available, computing a single maximal conflict-free set is enough. The selection of a single maximal subset is known as *maxichoice* and although it presents some incompatibilities with the AGM theory, it is not a problem to use such approach in the context of belief bases.

We focus specifically on partial meet contraction, in which we build maximally conflict-free sub-ontologies in a direct manner. We propose optimizations for the operators described in the literature and test their effectiveness on real world data gathered from the BioPortal repository[1].

The paper proceeds as follows: in the next Section, we present the operations for belief contraction and related work on applying belief change to ontologies. In Section 3, we present the algorithms and heuristics for the construction of the Remainder Set. In Section 4, we describe the experiments comparing our approach to the existing ones and in Section 5 we present our conclusions and discuss future work.

2 Belief Base Contraction

The operation of removing information from a knowledge base is called *contraction* and there are two main forms of doing so, *Kernel* contraction and *Partial Meet* contraction. We start by introducing the operations as they were originally described in the literature. A *belief base* is a set of formulas in a given logic.

The first operation works by constructing a set of minimal conflict preserving sub-bases, called *Kernel Set*, and then removing the least preferred (according to some preference order) sentences of each of the sub-bases. The operation of ranking axioms and selecting the least preferred one is called *Incision Function*. These concepts are formally described in Definitions 1 and 2 respectively. Both concepts are used in

[1] http://bioportal.bioontology.org/

order to formulate the operation for Kernel contraction presented in Definition 3.

Definition 1 (Kernel Set) [Hansson, 1994] *Let B be a belief base and α a sentence. A set B' is an element of $B \perp\!\!\!\perp \alpha$ iff B' is a minimal subset of B such that $B' \models \alpha$:*

- $B' \subseteq B$
- $B' \models \alpha$
- *If $B'' \subset B' \subseteq B$, then $B'' \not\models \alpha$ (minimality)*

Definition 2 (Incision Function) [Hansson, 1994] *Let B be a belief base. For any sentence α, an incision function for B is a function σ such that:*

- $\sigma(B \perp\!\!\!\perp \alpha) \subseteq \bigcup(B \perp\!\!\!\perp \alpha)$ *and*
- *if $\emptyset \neq X \in B \perp\!\!\!\perp \alpha$ then $X \cap \sigma(B \perp\!\!\!\perp \alpha) \neq \emptyset$*

Definition 3 (Kernel Contraction) [Hansson, 1994] *Let B be a belief base. For any sentence α and incision function σ, the kernel contraction of B by α is given by:*

- $B -_\sigma \alpha = B \setminus \sigma(B \perp\!\!\!\perp \alpha)$

Partial Meet contraction works by building maximal conflict-free subsets directly. The set of all possible maximal conflict-free subset is called *Remainder Set*. The number of elements in the Remainder Set can be possibly large, thus we need to select which ones suit best our needs. The operation of selecting the *best* elements from the remainder is called *Selection Function*. Both concepts are formally described in Definition 4 and 5, respectively. Definition 6 shows how these two concepts are used to build a Partial Meet contraction operator.

Definition 4 (Remainder Set) [Alchourron *et al.*, 1985] *Let B be a belief base and α a sentence. A set B' is an element of the remainder $B \perp \alpha$ if and only if it is a maximal subset of B that does not imply α:*

- *B' is a subset of B ($B' \subseteq B$)*
- $B' \not\models \alpha$
- *If $B' \subset B'' \subseteq B$, then $B'' \vdash \alpha$*

Definition 5 (Selection Function) [Alchourron *et al.*, 1985] *Let $\mathcal{L}$ be a language and B a belief base of this language. For any sentence α, a selection function for B is a function γ such that, for any sentence $\alpha \in \mathcal{L}$:*

- $\gamma(B \perp \alpha) \subseteq B \perp \alpha$
- *if $B \perp \alpha \neq \emptyset$, then $\gamma(B \perp \alpha) \neq \emptyset$*
- *if $B \perp \alpha = \emptyset$, $\gamma(B \perp \alpha) = \{B\}$*

Definition 6 (Partial Meet Contraction) [Alchourron *et al.*, 1985] *Let B be a belief base, α a sentence and γ a selection function. The partial meet contraction of B by α is given by:*

- $B -_\gamma \alpha = \bigcap \gamma(B \perp \alpha)$

Partial meet contraction is the most well-known construction for belief contraction. When Hansson introduced kernel base contraction in [Hansson, 1994], it was seen as a solution much closer to work in artificial intelligence, such as Truth Maintenance Systems [Doyle, 1979] and Model Based Diagnosis [Reiter, 1987; Wassermann, 2000] and it seemed much more amenable to implementation than partial meet contraction. Maybe for this reason, most tools for ontology debugging use something similar to kernels. Schlobach and Cornet [Schlobach and Cornet, 2003] call the minimal inconsistent subsets of an ontology *MUPS* (Minimal Unsatisfiability-Preserving Sub-terminologies) and the minimal incoherent subsets of an ontology *MIPS* (Minimal Incoherence-Preserving Sub-terminologies). They present an algorithm for computing all the minimal subsets of a given knowledge base that have a given consequence, which are called *MinAs* (Minimal Axiom set) in [Baader *et al.*, 2007] and served as the basis for Horridge's *Justifications* [Horridge, 2011].

Operators for building kernel elements for ontologies are largely described at the literature, for instance in [Kalyanpur, 2006; Suntisrivaraporn *et al.*, 2008; Kalyanpur *et al.*, 2007; Ji *et al.*, 2009; Horridge, 2011]. Several optimizations have also been proposed, like the *Sliding Window* technique [Kalyanpur, 2006] or the *Divide and Conquer* technique [Junker, 2001] or the *Syntactic Relevance* [Ji *et al.*, 2009; Huang *et al.*, 2005]. They all aim to reduce the number of calls to the reasoner mechanism and also to reduce the overall execution time. As we can see, the kernel building challenge is a problem highly studied on the literature.

On the other hand, Partial Meet contraction has not been given the same amount of attention. It has been mentioned in [Wassermann, 2000] and [Nyssen, 2009] that the Remainder Set can be obtained from the Kernel Set using Reiter's hitting sets algorithm [Reiter, 1987]. However, it is not easy to find works that implement a Partial Meet contraction directly. In [Resina *et al.*, 2014], the authors describe a procedure to build the Remainder Set, which was used as a baseline for the comparison of the heuristics we proposed.

3 Construction of the Remainder Set

The approach used to build a Remainder Set element is similar to the one used to build a Kernel Set element. For the Kernel, most algorithms are described in the form of an *expand-shrink* algorithm. The idea is to start from the empty set and add formulas to it until the conflict appears - the *expand phase* - and then, as the set may not be minimal, remove any formulas whose removal does not solve the conflict - the *shrink phase*.

In the case of the Remainder Set elements we use a shrink-expand algorithm [Ribeiro, 2013], that first removes axioms until the conflict no longer exists, then tries to add back the removed axioms in order to guarantee maximality. The basic algorithm for building a Remainder can be seen in Listing 1, where $\mathcal{O}$ is the ontology for which we want to find the Remainder Set and φ is the formula to be contracted, i.e., the algorithm returns an element of $\mathcal{O} \perp \varphi$.

Listing 1: Black-Box Algorithm for building a single element of the remainder

```
Black-box-remainder(O, φ):
 #Shrink first
 removed_elements← O
 remainder_element← ∅
 #Now Expand
```

```
for each α ∈ removed_elements do
  if(remainder_element ∪ {α} ⊭ φ) then
    remainder_element ←
                  remainder_element ∪ {α}
return remainder_element
```

The algorithm presented in Listing 1 was proposed by Resina et al. in [Resina *et al.*, 2014]. The author bypasses the shrink phase by removing all the axioms from $\mathcal{O}$ at once (trivial shrinking) and passes to the expand, which is done iteratively. A small variation of this algorithm would use an iterative strategy for both the expansion and the shrink phase. In the rest of this Section, we will describe a set of heuristics that we developed with the goal of optimizing the execution of the classical algorithm. The heuristics here proposed are inspired by the ones used to optimize the classical black-box algorithm for building kernel elements and are grouped according to the phase in which they are used.

3.1 Optimizations for the Shrink Phase

The first heuristics we propose follows the same principle of the Sliding Window proposed by Kalyanpur in [Kalyanpur *et al.*, 2005]. The idea is that we can optimize the shrink phase if we, at each interaction remove more than one axiom at a time. If the removed axiom set fixes the ontology conflict, then we can move on to the expand phase and try to re-add the removed axioms one by one. The algorithm for such heuristics can be seen in Listing 2.

Listing 2: Sliding Window Algorithm for Remainder

```
Sliding_Window(O,φ, window_size):
 remainder_set ← ∅
 window_start ← 1
 window_end ← window_size
 O_list ← O
 while window_start <= |O_list| do
   O_sublist ←
        O_list.subList(window_start,window_end)
   remainder_set ← remainder_set ∪ O_sublist
   if remainder_set ⊨ φ then
     remainder_set ← remainder_set\O_sublist
     window_start ← window_start+1
   else
     window_start ← window_start+window_size
   window_end ← window_start+window_size
   if window_end > |O_list| then
     window_end ← |O_list|
 return remainder_set
```

Example 1 shows how the proposed heuristics works. Notice that this heuristics, as well as most of the algorithms in the literature, assume that the formulas are presented as a list, and therefore, ordered. In this work, as in [Horridge, 2011], we are not presuming any particular ordering, although we know that it affects the performance of the algorithms.

Example 1 *Consider the ontology* $\mathcal{O} = \{A \sqsubseteq B, B \sqsubseteq D, A \sqsubseteq E, E \sqsubseteq F\} \models A \sqsubseteq D$ *nd feed it to the algorithm in Listing 2.*

The algorithm then positions the window over the first axiom we would get:

$$\mathcal{O} = \{\boxed{A \sqsubseteq B, B \sqsubseteq D, A \sqsubseteq E}, E \sqsubseteq F\}$$

The sub-ontology resulting from removing the axioms inside the sliding window can still entail $A \sqsubseteq D$*, thus we have to slide the window to its next position, resulting in:*

$$\mathcal{O} = \{A \sqsubseteq B, \boxed{B \sqsubseteq D, A \sqsubseteq E, E \sqsubseteq F}\}$$

Now, after removing the axioms inside the window we remove the unwanted entailment $A \sqsubseteq D$*. The algorithm returns the subset removing the axioms within the sliding window.*

The next heuristics we propose is to start the shrinking first removing the axioms that share concepts in their signatures with the axiom causing the conflict. Listing 3 shows the algorithm that implements such heuristics. The algorithm relies on the function `Signature` that returns a set of all Classes used on the axiom.

Listing 3: Syntactic Conectedness based algorithm

```
Syntatic_Relevance_Black-box(O,φ):
 removed_elements← ∅
 #Shrink
 φ_sig← Signature(φ)
 for each α ∈ O do
   if φ_sig ∩ Signature(α) <> ∅ then
     removed_elements ← removed_elements∪{α}
 if O\ removed_elements ⊨ φ
   #Perform Trivial Shrink
   removed_elements← O
 #Expand ...
```

3.2 Optimizations for the Expand Phase

The heuristics that we propose for the expansion phase is based on the idea from Junker in [Junker, 2001]. The author proposes to use a divide and conquer based strategy to build explanations (kernel elements) for a given entailment. This is the same heuristics used by the Protégé OWLExplanation plugin.[2]

Our version uses the same idea, with a slight modification in the way we divide the input. In [Junker, 2001], the author divides the input in quarters and tries to combine such quarters hoping to obtain a conflicting subset, then the routine is executed recursively. In our case, we try to build the largest conflict-free subset, thus we only divide the input in halves and check whether any half is conflict free, then we apply the routine recursively on the remaining elements.

The algorithm for this heuristics can be seen in Listing 4.

Listing 4: Divide and Conquer based algorithm

```
Divide_and_Conquer_Remainder(O_list,φ, O_safe):
 if |O_list| = 1 then
   if O_list ∪ O_safe ⊭ φ then
     return O_list ∪ O_safe
   else
     return O_safe
 middle ← 1+|O_list|/2
 O_sublist ← O_list.sublist(1,middle)
 if(O_sublist ∪ O_safe ⊭ φ) then
   O_safe ← O_safe ∪ O_sublist
```

[2] https://github.com/matthewhorridge/owlexplanation

```
    return Divide_and_Conquer_Remainder(
        O_list \ O_sublist, φ, O_safe)
  else
    O_sublist ← O_list.sublist(middle, |O_list|)
    if(O_sublist ∪ O_safe ⊭ φ) then
      O_safe ← O_safe ∪ O_sublist
      return Divide_and_Conquer_Remainder(
          O_list \ O_sublist, φ, O_safe)
    else
      O_sublist ← O_list.sublist(1,middle)
      O_safe ← O_safe ∪
        Divide_and_Conquer_Remainder(O_sublist,
          φ, O_safe)
      O_sublist ← O_list.sublist(middle, |O_list|)
      return Divide_and_Conquer_Remainder(
          O_sublist, φ, O_safe)
```

Example 2 illustrates the execution of the algorithm.

Example 2 *Consider the ontology $\mathcal{O} = \{A \sqsubseteq B, B \sqsubseteq D, A \sqsubseteq F, F \sqsubseteq D\}$ and suppose that the entailment $\varphi = A \sqsubseteq D$ is conflicting with the purposes for which your ontology is being built.*

The first interaction of the algorithm would divide $\mathcal{O}$ into $\mathcal{O}_1 = \{A \sqsubseteq B, B \sqsubseteq D\}$ and $\mathcal{O}_2 = \{A \sqsubseteq F, F \sqsubseteq D\}$, then we need to check if any of the sub-ontologies is conflict free. It is the case for $\mathcal{O}_2$, thus a recursive call is done using the rest as input, i.e., $\mathcal{O}_1$.

The input $\mathcal{O}_1$ is then divided in $\mathcal{O}_{11} = \{A \sqsubseteq B\}$, from which we cannot infer $A \sqsubseteq D$. This is the very base case of the recursion (the base cannot be divided any further) so the algorithm tries to add it to the safe knowledge base $\mathcal{O}_2$ and the result is still safe, so a recursive call is done on the other half, i.e., $\mathcal{O}_{12} = \{B \sqsubseteq D\}$, as input. Again we reach the base case for our recursion, only this time we cannot add it because such an action would make the conflicting entailment valid again.

4 Experiments

We developed a series of software experiments to check the performance of the algorithms and heuristics that we proposed. These components were developed using the Java programming language and are available as a free/open source software at (`https://github.com/raphaelmcobe/ontology-debug-and-repair`). We used two datasets for our experiments. The first was automatically generated and the second used the data available at the BioPortal.

The idea behind these experiments was to verify if there was any case - even an artificially generated one - in which building the Remainder Set could be *easier* than building the Kernel Set. Thus, we were not concerned with the quality of the remainder element built since it has been shown in [Booth *et al.*, 2011] that it is always possible to derive the remainder set from the kernel set.

If we managed to find such input, we would like to verify if such cases could be found in real world ontologies. This would serve both as a redeemer of the partial meet construction in the Artificial Intelligence community and as evidence for building efficient ontology repair tools.

4.1 Experiment Design

The experiment aimed to prove three hypotheses:

H1: the heuristics proposed have a positive impact on the performance of the Remainder Set finding algorithm. We prove this hypothesis by measuring the overall impact on the execution time as well as the number of reasoner calls.

H2: there are cases at which building a single element of the Remainder Set is at least as efficient as building a single element of the Kernel Set. This hypothesis is proved if we are able to find cases at which building a remainder element is faster or makes less calls to the reasoner than building a kernel element.

H3: there are cases at which it is easier to build a remainder element than building the whole Kernel Set. We verify with this hypothesis that that even if we are not able to build a remainder element as fast as a kernel element it may be the case that building the whole Kernel Set takes longer or makes more calls to the reasoner than building a single remainder element. This is still a good result because building the Kernel Set is just the first step towards fixing the conflicts. The user will still have to use an incision function to order and remove the less preferred axioms of each kernel element. In this case we were not concerned with the quality of the remainder element built.

For our experiment we chose to collect two metrics: **Reasoner calls**, and **Overall execution time**. We tested the following combination of heuristics for the remainder element building:

R_1**:** Syntactic Connectedness based Shrinking and Iterative Expansion;

R_2**:** Syntactic Connectedness based Shrinking and Divide and Conquer based Expansion;

R_3**:** Trivial Shrinking (removal of all ontology axioms) and Divide and Conquer based Expansion;

R_4**:** Trivial Shrinking and Iterative Expansion;

R_5**:** Sliding Window Shrinking and Iterative Expansion;

R_6**:** Iterative Shrinking and Iterative Expansion.

For the kernel element building we used the combination of Syntactic Connectedness based Expansion and Divide and Conquer based Shrinking. This combination is the same used by Horridge in [Horridge, 2011] and in the OWLExplanation plugin, which is considered state of art implementation for building justifications (Kernel Sets). The author argued that such heuristics presented better performance. For other optimizations refer to [Horridge, 2011]. The plugin was used in the experiments described in Section 4.3 where we build the whole Kernel Set.

The algorithms presented on this paper were implemented using the Java Language (Oracle JVM v. 1.7.0_51) and the tests were executed on a Linux machine running Ubuntu 13.10 (Kernel 3.11.0-19-generic) with an Intel 12 core CPU (XEON X5660) and 32 GB of RAM. The java VM was fed with the parameters: `-Xms 2048m -Xmx16384m`. Each

test had a timeout, arbitrarily defined of 45 seconds for the tests with generated data and 330 seconds for each entailment tested at the BioPortal ontologies. The version of the OWLExplanation plugin used was 1.0.3, checked out from its github repository on October 14th, 2014.

4.2 Generated Data

The generated data used to test our hypotheses follows Definition 7.

Definition 7 *[Resina* et al.*, 2014] A Large Kernel and Small Remainder Ontology is built using the template:*

$$\mathcal{O} = \left\{ \begin{array}{c} (B \sqcup B')(a) \\ B \sqcup B' \sqsubseteq B_1, B \sqcup B' \sqsubseteq B'_1, \\ B_1 \sqcup B'_1 \sqsubseteq B_2, B_1 \sqcup B'_1 \sqsubseteq B'_2, \\ \vdots \\ B_{n-1} \sqcup B'_{n-1} \sqsubseteq B_n, B_{n-1} \sqcup B'_{n-1} \sqsubseteq B'_n \end{array} \right\}$$

It is interesting to notice that if we build the Kernel Set $\mathcal{K}$ and the Remainder Set $\mathcal{R}$ for the entailment $(B_n \sqcup B'_n)(a)$ in $\mathcal{O}$ we notice that $|\mathcal{K}| = 2^n$ and $|\mathcal{R}| = n + 1$. Example 3 shows how Definition 7 can be used to generate an ontology.

Example 3 *If we build the ontology $\mathcal{O}$ using the template described at the Definition 7 and calculate the kernel $\mathcal{K}$ and remainder $\mathcal{R}$ sets, considering $n = 2$ and the entailment $\mathcal{O} \models (B_2 \sqcup B'_2)(a)$ we would get:*

$$\mathcal{K} = \left\{ \begin{array}{l} \{(B \sqcup B')(a), B \sqcup B' \sqsubseteq B_1, B_1 \sqcup B'_1 \sqsubseteq B_2\}, \\ \{(B \sqcup B')(a), B \sqcup B' \sqsubseteq B'_1, B_1 \sqcup B'_1 \sqsubseteq B_2\}, \\ \{(B \sqcup B')(a), B \sqcup B' \sqsubseteq B_1, B_1 \sqcup B'_1 \sqsubseteq B'_2\}, \\ \{(B \sqcup B')(a), B \sqcup B' \sqsubseteq B'_1, B_1 \sqcup B'_1 \sqsubseteq B'_2\} \end{array} \right\}$$

$$\mathcal{R} = \left\{ \begin{array}{c} \mathcal{O} \setminus \{(B \sqcup B')(a)\}, \\ \mathcal{O} \setminus \{B \sqcup B' \sqsubseteq B_1\}, \\ \mathcal{O} \setminus \{B \sqcup B' \sqsubseteq B'_1\} \end{array} \right\}$$

Figures 1 and 2 sumarize the comparison between operators for building a kernel and a remainder element. We chose the best results among the heuristics combinations for building remainder elements. In the case of the remainder building algorithm we selected the two best results: the one using the Syntactic Connectedness for the shrinking and iterative expansion and the Divide and Conquer for expansion and Trivial shrinking. We also plotted the results of the baseline algorithm for building a remainder element, proposed by Resina et al. [Resina *et al.*, 2014], which uses the trivial shrink and iterative expansion combination.

We can see that the operators for building a remainder element had a better performance for both algorithms in the case of reasoner calls. The same could be observed during the time analysis. In this case, we observed a better performance by the remainder building algorithm that used syntactic connectedness, followed by the kernel building algorithm and the worst result was the remainder building algorithm that used Divide and Conquer.

We can see from the Figures that we were able to prove all of our hypotheses, we were able to introduce heuristics that resulted in better performance than what is described at the literature (H1). Our results also showed that, for this kind

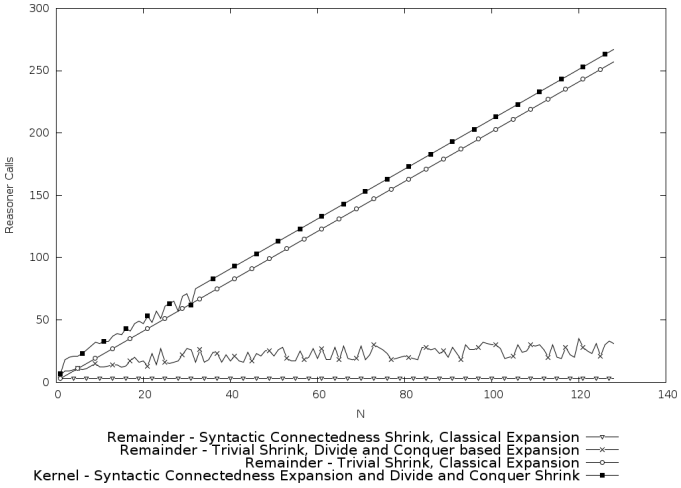

Figure 1: Comparison between algorithm for building a kernel and remainder element (Reasoner calls)

of input, building a remainder element is easier - in terms of reasoner calls and overall execution time - than building a kernel element (H2 and H3).

We were able to find a template for constructing an ontology for which building a remainder element is easier than building a kernel element. We went a step further and tried to find real world ontologies at which that is also the case.

During the development of our study we also evaluated experiments using ontologies with small Kernel Sets and large Remainder Sets. Such dataset only confirmed the intuition that building a kernel element has better performance than building a remainder element.

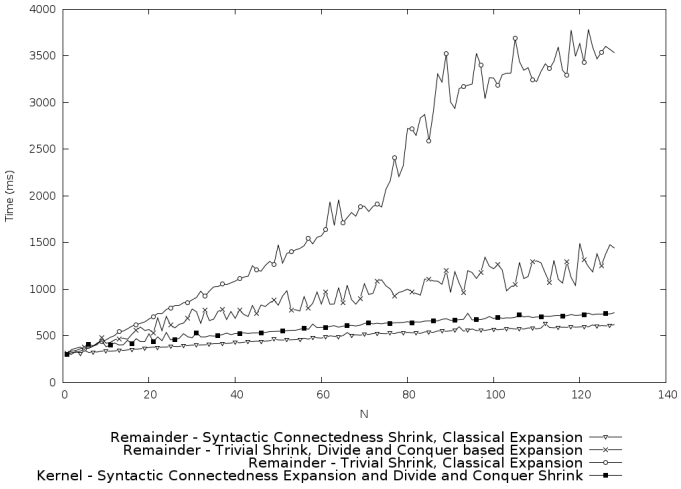

Figure 2: Comparison between algorithm for building a kernel and remainder element (Execution time)

4.3 BioPortal Data

We have conducted an experiment using the data available at the BioPortal repository, maintained by the *National Center for Biomedical Ontology* - NCBO. The same dataset was

used by Horridge at [Horridge, 2011]. The expressivity of the ontologies collected vary from $\mathcal{EL}$ to $\mathcal{SHROIQ}$.

During the data selection procedure, Horridge filtered only ontologies that were described using $\mathcal{SHROIQ}$ or a less expressive language. Then he filtered ontologies that were written in OWL and OBO since standard tools could be used to manipulate them. After selecting the files, Horridge enumerated all *non-trivial* entailments, which are axioms entailed by the reasoner that were not explicitly defined. The resulting dataset had 72 ontologies.

We selected 51 of the 72 ontologies to test our algorithms. We chose the ontologies that had 1 MB or less, since our experiments were not yet optimized and do not scale. The total amount of tested entailments was 3812.

We gathered data regarding the overall execution time and reasoner calls and only compared the execution of the algorithm for finding a remainder element and the whole kernel, provided by the OWLExplanation plugin.

During the experiment we used the same heuristics combination from the experiment with generated data for the remainder element building algorithm. For the comparison, we tested the state of art kernel finding software, which is distributed with the Protégé[3] editor.

Each ontology had its own set of accompanying entailments to be tested and in Table 1 we show the percentage in which the algorithm behaved better, i.e., the percentage of entailments for which the operator had the best performance.

For the case of the reasoner calls test we observed that, within the defined timeout, 19 ontologies had better performance on building a single remainder element compared to building the whole Kernel Set. That represents approximately 40% of the ontologies. This result helped us prove our third hypothesis (H3) with real world data at least comparing the number of reasoner calls. Also, in all cases, the heuristics that we proposed had better performance than the baseline algorithm for building a remainer element by Resina et al. [Resina *et al.*, 2014], with exception of the Sliding Window heuristic, that had worse performance than the algorithm from Resina et al. [Resina *et al.*, 2014]. We believe that this is the case because we have to better calibrate the sliding window size before running the experiments. We used 10 as the window size. This size was the same reported by Kalyanpur [Kalyanpur, 2006] as being the best for the kernel element building algorithm based on sliding windows. This result goes on the direction of proving our first hypothesis (H1) also with real world data.

Table 1 summarizes the data collected at our experiment regarding the number of reasoner calls.

The results comparing execution time were not as good as the number of reasoner calls. The numbers for this experiment are presented on Table 1.

Operator	**Total** RC	**%** - RC	**Total** Time	**%** - Time
R_1	1517	39.80%	94	2.47%
R_2	22	0.58%	1	0.03%
R_3	51	1.34%	41	1.08%
R_4	2	0.05%	2	0.05%
R_5	1	0.03%	1	0.03%
R_6	5	0.13%	2	0.05%
OWLExplanation Plugin	1970	51.58%	3427	89.90%

Table 1: Total number of entailments and percentages at which each algorithm presented better performance results: reasoner calls - RC, and execution time - time

5 Final Remarks

In this paper we presented new algorithms for building remainder elements. We conducted experiments in order to prove that in some (real) cases it is easier to build a remainder element than the whole kernel. We believe that building the whole kernel can be used as a fine-grained debugging strategy and although it is the case that it is easier to build the Kernel Set in general, the user may still have to define an incision function in order to fix the ontology. Such function might cause an overhead. We were not specially concerned with the remainder element quality. Still, the Syntactic Connectedness heuristics can build remainder elements discarding the axioms syntactically connected to the unwanted entailment first, which can be a desired property at the ontology repair.

We tested our heuristics using real and automatically generated data. We have found real world cases that proved that it is not the case that it is always easier to build the Kernel Set than building a remainder element. Still, further investigation is needed in order to check how often these cases appear in larger data sets.

During our experiment we observed a few interesting cases (6.4% of the entailments), where the kernel algorithm could not terminate within the allocated time: ontology semanticscience-integrated-ontology (3 entailments), imgt-ontology (25 entailments), biotop (4 entailments), amino-acid (3 entailments), and adverse-event-reporting-ontology (1 entailment).

In ontology *imgt-ontology*, the Syntactic Connectedness heuristics did not present the best performance results. In this case, the trivial shrinking proved to be a better option in 3 entailments checked.

For the future, we intend to test our algorithms with a larger dataset of real world ontologies and implement a Protégé plugin for debugging based on remainder elements. We also plan to check the algorithms behaviour after introducing the idea of ontology modules. In that way, we could isolate sub-ontologies, preferring a sub-set of axioms over another, thus building remainder elements more suitable for the ontology designer needs.

References

[Alchourron *et al.*, 1985] C.E. Alchourron, P. Gärdenfors, and D. Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *The Journal of Symbolic Logic*, 50(2):510–530, 1985.

[Baader *et al.*, 2007] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the descrip-

[3] http://protege.stanford.edu/

tion logic EL. In *Proceedings of the International Workshop on Description Logics (DL)*, 2007.

[Booth *et al.*, 2011] Richard Booth, Thomas Meyer, Ivan Varzinczak, and Renata Wassermann. On the link between partial meet, kernel, and infra contraction and its application to horn logic. *J. Artif. Int. Res.*, 42(1):31–53, September 2011.

[Doyle, 1979] Jon Doyle. A truth maintenance system. *Artificial Intelligence Journal*, 12(3):231–272, 1979.

[Flouris, 2006] Giorgos Flouris. *On Belief Change and Ontology Evolution*. PhD thesis, University of Crete, 2006.

[Haase *et al.*, 2005] P. Haase, F. van Harmelen, Zh. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies. In *Proceedings of the Fourth Internation Semantic Web Conference*, volume 3729 of *LNCS*, pages 353–367. Springer, 2005.

[Hansson, 1994] Sven Ove Hansson. Kernel contraction. *The Journal of Symbolic Logic*, 59(03):845–859, 1994.

[Hansson, 1999] Sven Ove Hansson. *A Textbook of Belief Dynamics*. Kluwer Academic Press, 1999.

[Horridge, 2011] Matthew Horridge. *Justification based explanation in ontologies*. PhD thesis, the University of Manchester, 2011.

[Huang *et al.*, 2005] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 454–459, 2005.

[Ji *et al.*, 2009] Qiu Ji, Guilin Qi, and Peter Haase. A relevance-directed algorithm for finding justifications of dl entailments. In Asuncin Gmez-Prez, Yong Yu, and Ying Ding, editors, *The Semantic Web*, volume 5926 of *Lecture Notes in Computer Science*, pages 306–320. Springer Berlin Heidelberg, 2009.

[Junker, 2001] Ulrich Junker. QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms. In *Proceedings of the IJCAI Workshop on Modelling and Solving Problems with Constraints (IJCAI'01)*. Morgan Kaufmann, 2001.

[Kalyanpur *et al.*, 2005] Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 1(1):36–49, 2005.

[Kalyanpur *et al.*, 2007] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2007.

[Kalyanpur, 2006] Aditya Anand Kalyanpur. *Debugging and repair of OWL ontologies*. PhD thesis, the University of Maryland, 2006.

[Nyssen, 2009] Rafael Peñaloza Nyssen. *Axiom pinpointing in description logics and beyond*. PhD thesis, Dresden University of Technology, 2009.

[Qi *et al.*, 2008] Guilin Qi, Peter Haase, Zhisheng Huang, Qiu Ji, JeffZ. Pan, and Johanna Vlker. A kernel revision operator for terminologies algorithms and evaluation. In *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2008.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Resina *et al.*, 2014] Fillipe Resina, Márcio Moretto Ribeiro, and Renata Wassermann. Algorithms for multiple contraction and an application to OWL ontologies. In *Proceedings of the Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, 2014.

[Ribeiro, 2013] Márcio Moretto Ribeiro. *Belief revision in non-classical logics*. Springer, 2013.

[Schlobach and Cornet, 2003] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 355–362. Morgan Kaufmann, 2003.

[Schlobach *et al.*, 2007] S. Schlobach, Z. Huang, R. Cornet, and F. van Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39:317–349, 2007. 10.1007/s10817-007-9076-z.

[Suntisrivaraporn *et al.*, 2008] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A modularization-based approach to finding all justifications for OWL DL entailments. In John Domingue and Chutiporn Anutariya, editors, *The Semantic Web*, volume 5367 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2008.

[Wassermann, 2000] Renata Wassermann. An algorithm for belief revision. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR)*. Morgan Kaufmann, 2000.