Detecting Hidden Errors in an Ontology Using Contextual Knowledge

Mehdi Teymourlouie[a], Ahmad Zaeri[a], Mohammadali Nematbakhsh[a,*], Matthias Thimm[b], Steffen Staab[b]

[a]*Department of Software Engineering, Faculty of Computer Engineering, University of Isfahan, Iran.*
[b]*Institute for Web Science and Technologies, Universität Koblenz-Landau, Germany.*

Abstract

Due to modeling errors in designing ontologies, an ontology may carry incorrect information. Ontology debugging can be helpful in detecting errors in ontologies that are increasing in size and expressiveness day by day. While current ontology debugging methods can detect logical errors (incoherences and inconsistencies), they are incapable of detecting hidden modeling errors in coherent and consistent ontologies. From the logical perspective, there are no errors in such ontologies, but this study shows some modeling errors may not break the coherency of the ontology by not participating in any contradiction. In this paper, contextual knowledge is exploited to detect such hidden errors. Our experiments show that adding general ontologies like DBpedia as contextual knowledge in the ontology debugging process results in detecting contradictions in ontologies that are coherent.

Keywords: Ontology Debugging, Hidden Modeling Errors, Contextual Knowledge, Incoherency, Inconsistency

1. Introduction

Ontologies play a main role in establishing the *Semantic Web* by providing meaning to the information published on the *Web of Data* (Bizer et al., 2009). Many ontologies have been developed to model the real-world concepts and the relations between them. Due to the modeling errors occurring in the design of ontologies, an ontology may carry incorrect information. Also, reasoning and answering queries in these ontologies result in incorrect information extraction. Debugging these faulty ontologies by human experts can be difficult as the size and expressiveness of ontologies are increasing day by day.

*Corresponding author. Address: Hezar-Jerib Ave. Isfahan 81746-73441, Iran. Phone / Fax :(+98) 313 793 4106

Email addresses: m.teymourlouie@eng.ui.ac.ir (Mehdi Teymourlouie), zaeri@eng.ui.ac.ir (Ahmad Zaeri), nematbakhsh@eng.ui.ac.ir (Mohammadali Nematbakhsh), thimm@uni-koblenz.de (Matthias Thimm), staab@uni-koblenz.de (Steffen Staab)

Errors in ontology can be divided into two categories: syntactical and semantic errors. Syntactical errors occur when the ontology is not compatible with the syntax or format of the intended ontology language. Semantic errors occur when there are some unintended meanings that could be concluded from an ontology while they are not true in the modeled domain. While syntactic errors can be detected and solved by most ontology editing tools (*e.g.* Protégé[1]), semantic errors are hard to resolve. Finding automatic methods to debug ontologies has received much attention in the last few years (Jannach et al., 2016; Arif et al., 2016; Papacchini & Schmidt, 2015; Friedrich, 2014; Stuckenschmidt, 2008; Moodley, 2010; Wang & Xu, 2008; Rodler et al., 2013; Shchekotykhin et al., 2012; Corcho et al., 2009; Schlobach et al., 2007; Stuckenschmidt, 2013; Ji et al., 2012; Kalyanpur et al., 2005; Bell et al., 2007; Lehmann & Bühmann, 2010; Roussey & Zamazal, 2013).

Previous works focused on detecting logical contradictions (*i.e.* incoherences and inconsistencies) as symptoms of the existence of modeling errors (Meilicke & Stuckenschmidt, 2008). These methods search for chains of axioms that preserve a logical contradiction. Such logical contradictions are then eliminated by detecting and repairing the faulty axioms, which are the root causes of contradictions, among the detected chains. In these methods, debugging stops when no sign of modeling errors (incoherency or inconsistency inside the ontology) remains.

A major drawback of the current methods is that they do not consider all kinds of modeling errors in the ontology and only try to detect incoherences and inconsistencies. Hence, in these approaches, the problem of debugging incorrect information is reduced to eliminating incoherences and inconsistencies. However, an ontology can be coherent and consistent but still include modeling errors.

From the ontology debugging perspective, we can divide the ontology axioms via the categorization shown in Figure 1. Axioms that are part of some contradiction are called *Suspected Axioms*. These axioms could be correct or faulty. The process of detecting faulty axioms among Suspected Axioms is discussed briefly in Section 3.3. Those axioms which are not part of any contradiction are called *Free Axioms* (Hunter & Konieczny, 2010). Previous works on ontology debugging considered free axioms as correct axioms because they are not taking part in any contradiction and debuggers could not find any evidence of incorrectness among them.

In this paper, we go one step deeper and divide free axioms into two separate categories: 1) *truly correct axioms* which are truly correct and match the real world modeled in the ontology; and 2) *hidden error axioms* which are incorrect axioms regarding the real world. Hidden error axioms could be considered as implicit errors as opposed to explicit errors that are found by internal conflict checking of the ontologies.

The main focus of this work is to distinguish hidden errors from truly correct axioms. The goal is to construct some contradictions using hidden errors and some external background knowledge. In this way, hidden errors will take part in some new contradiction and they will not be free axioms anymore. We assume that the background knowledge is correctly representing the real world. Hence, conflicting with background knowledge shows some conflict with the real world. Thus, the debugger

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

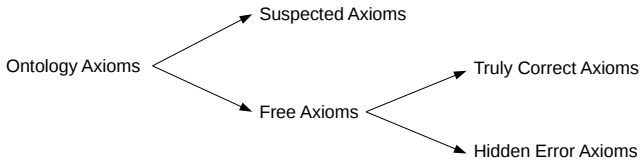

Figure 1: The Category of Axioms Types from Ontology Debugging Perspectives.

could consider those axioms that are conflicting with background knowledge as suspected axioms.

Consider the following ontology given in some description logic (Baader et al., 2010) :

$$Penguin \sqsubseteq Bird \tag{1}$$
$$Bird \sqsubseteq FlyingAnimals \tag{2}$$

This ontology is coherent and consistent, so there is no logical contradiction in the ontology and current methods cannot detect any incorrect information. Axioms 1 and 2 are free axioms. It is inferred from this ontology that *Penguin* is a subclass of *FlyingAnimal*. But we know that in reality penguins cannot fly. Thus, there should be some error in this ontology. Since the errors are not part of any logical contradiction, they are *hidden errors*. The question is how can we detect these hidden errors?

If we ask a human user to debug this ontology, the user can evaluate the ontology with his/her background knowledge. If there is some statement in the user's background knowledge that says "Penguins cannot fly", then the user can detect some contradiction between the information in the ontology and his/her own knowledge. Hence, the incorrectness of the information can be concluded.

From a technical point of view, an ontology can be coherent and consistent internally, but if this information is evaluated *w.r.t.* to other knowledge-bases, conflicts may arise. For example, if we evaluate the above ontology with an ontology containing Axioms (3) and (4), it will become incoherent and Axioms 1 and 2 will be part of the contradiction. Thus, they will be marked as suspected axioms and the error could be detected in the debugging process.

$$Penguin \sqsubseteq NonFlyingAnimals \tag{3}$$
$$NonFlyingAnimals \sqsubseteq \neg FlyingAnimals \tag{4}$$

In this work, we develop an approach using contextual knowledge to debug ontologies. As many errors may be hidden inside the coherent or consistent ontologies, we want to develop methods to utilize the knowledge of other knowledge-bases for debugging modeling errors of ontologies. Hence, the goal of our research is to detect more contradictions by adding and aligning more background knowledge to ontology debugging and making hidden error axioms as suspect axioms.

The contributions of this work could be summarized as:

1. To the best of our knowledge, this is the first work that outlines the importance of checking the correctness of ontology with respect to other knowledge bases.

We show that even coherent ontologies may contain some hidden errors which could not be detected by internal coherence checking.

2. Proposing an approach to detect hidden errors is the other contribution of this paper. We examine the effect of adding background knowledge to the ontology debugging compared with internal incoherency checking used in the common ontology debugging methods.
3. Three new functions introduced to evaluate contradictions. These functions are used to detect error axioms among suspected axioms.

This paper is organized as follows. Section 2 provides basic terminologies in the ontology debugging field. In Section 3, hidden error detection is discussed in detail. Experiments to show the effectiveness of our proposed method in debugging real ontologies are described in Section 4. Section 5 reviews the related works and finally our conclusions are drawn in Section 6.

2. Preliminaries of Ontology Debugging

In the rest of this paper, we assume that the readers are familiar with Description Logics (DL), which is a family of First Order logics. DL is the underlying logic employed to formalize the representation of concepts and their relations in the ontologies. See the respective handbook (Baader et al., 2010) for a detailed description of DL.

As stated by Gruber (1993), an ontology is "a formal specification of a shared conceptualization". An Ontology is a way of knowledge representation to model specific domain of knowledge. An Ontology is formally defined by Definition 1.

Definition 1 (Ontology)**.** *An ontology $\mathcal{O}$ is a set of axioms, having Signature of $\mathcal{S}_\mathcal{O}$*

$$\mathcal{O} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$$
$$\mathcal{S}_\mathcal{O} =< C_\mathcal{O}, R_\mathcal{O}, I_\mathcal{O} >$$

where $C_\mathcal{O}$ is a set of Concepts, $R_\mathcal{O}$ is a set of Relations (Properties) and $I_\mathcal{O}$ is a list of Individuals represented in the ontology.

Usually, ontologies are divided into two parts: Terminological Box (TBox) and Assertional Box (ABox). The TBox includes axioms' defining concepts of the domain and the relations between them. The ABox includes axioms asserting the individuals into concepts and relations defined in the TBox.

Ontologies are created manually by human experts or automatically by applying some ontology learning methods. Both of the methods are prone to errors. When speaking about errors, we are talking about the semantic errors occurring during the modeling process rather than syntactical errors. Nowadays detecting and fixing syntactical errors by ontology editors is fairly easy.

One of the most common semantic errors in ontology modeling is incoherency. Incoherent ontology is defined by Definition 2 which is an ontology with at least one unsatisfiable concept.

Definition 2 (Incoherent Ontology)**.** *Ontology $\mathcal{O}$ is incoherent iff $\exists C \in C_\mathcal{O}$ $s.t.$ $\mathcal{O} \models C \sqsubseteq \bot$.*

Unsatisfiable concepts (classes) in an ontology are usually unintended results of some erroneous axioms. The *Theory of Diagnosis* developed by Reiter (1987) states that when there is some unwanted behavior in a system, there should be some parts in the system that are malfunctioning. So considering an ontology as a system and unsatisfiable classes as the unwanted behavior of ontology, then there should be some axioms in the ontology that are incorrectly modeled. Thus, the task of ontology debugging is to detect such incorrect axioms.

To explain why a concept C is unsatisfiable, one can detect a set of axioms in the ontology which preserve the unsatisfiability of it. These sets are called Minimum Unsatisfiability Preserving Set (MUPS). MUPS is defined formally as Definition 3.

Definition 3 (MUPS)**.** *Minimum Unsatisfiability Preserving Set:*

$$\mathcal{M}(C) = \{\mathcal{M} | \mathcal{M} \subseteq \mathcal{O}, \mathcal{M} \models C \sqsubseteq \bot, \forall \mathcal{M}' \subset \mathcal{M}, \mathcal{M}' \not\models C \sqsubseteq \bot\}$$
$$\mathcal{M}(\mathcal{O}) = \bigcup_{C \in C_{\mathcal{O}}} \mathcal{M}(C)$$

Definition 3 defines an MUPS as a minimum set of all the axioms which altogether imply that C is unsatisfiable. For an unsatisfiable concept C, it could be more than one MUPS. So, $\mathcal{M}(C)$ is the set of all MUPSs which imply that C is unsatisfiable. From Definition 3, we can conclude that if concept C is satisfiable then the set of MUPSs will be empty (*i.e.* $\mathcal{M}(C) \neq \emptyset$ iff C is unsatisfiable in $\mathcal{O}$.) and if an ontology is coherent, there will be no MUPS for its concepts (*i.e.* $\mathcal{M}(\mathcal{O}) = \emptyset$ iff $\mathcal{O}$ is coherent.).

We know that in each MUPS there is at least one incorrect axiom which causes the unsatisfiability of at least one concept. For eliminating each MUPS, a diagnosis should be found. Definition 4 defines Diagnosis as a set of axioms wherein removing them from ontology results in coherency of the ontology. In fact, a diagnosis is a hitting set for $\mathcal{M}(\mathcal{O})$.

Definition 4 (Diagnosis)**.** $D \subseteq \mathcal{O}$ *is a diagnosis for incoherent ontology* $\mathcal{O}$ *iff* $\mathcal{M}(\mathcal{O} \setminus D) = \emptyset$*. The set of Diagnosis for ontology* $\mathcal{O}$ *will be denoted as* $\mathcal{D}(\mathcal{O})$.

Hence, the task of ontology debugging can be summarized as detecting a set of axioms that are incorrect, eliminating them from the ontology, and turning the ontology back to the coherent state.

3. Detecting Hidden Errors

In Section 1, we discussed that there are some incorrect axioms in ontologies that are not part of any MUPS. We called them *hidden errors*. Finding these hidden errors as well as the other incorrect axioms is the goal of our approach.

The idea behind adding background knowledge to the debugging process is that no ontology has a complete representation of its modeled domain. Therefore, error axioms could be hidden when some knowledge is missed in the ontology. Thus, if we add some correct knowledge to the ontology, we can help debugging methods detect some new MUPSs, which include those hidden erroneous axioms.

Different ontologies modeling the same domain are good candidates for such contextual background knowledge. They model the same domain from different perspectives and may carry some knowledge that is not in the ontology being debugged. Our approach is to use such ontologies in the debugging process. Detecting hidden errors could be possible by detecting new MUPSs that could have not been found without adding the contextual knowledge. Before we start explaining our suggested process, let's define the background contextual knowledge.

Definition 5 (Knowledge-Base Profile)**.** *A Knowledge-Base Profile is a set of ontologies used as background knowledge. Hence, profile ontology consists of the union of axioms in all of the ontologies in a knowledge-base profile:*

$$\begin{aligned}\mathcal{PROFILE} &= \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_n\} \\ \mathcal{O}_{profile} &= \bigcup_{i=1}^{m} \mathcal{O}_i \\ \mathcal{S}_{\mathcal{O}_{profile}} &=< C_{\mathcal{O}_{profile}}, R_{\mathcal{O}_{profile}}, I_{\mathcal{O}_{profile}} > \\ &=< \bigcup_{i=1}^{m} C_i, \bigcup_{i=1}^{m} R_i, \bigcup_{i=1}^{m} I_i >\end{aligned}$$

We call the background ontologies the *Knowledge-Base Profile*, which is a set of ontologies modeling the same domain as the ontology being debugged. Profile ontology ($\mathcal{O}_{profile}$) is the ontology consisting of the union of all the axioms in the knowledge-base profile ontologies.

The suggested process is depicted in Figure 2. The first step is to merge the profile ontology ($\mathcal{O}_{profile}$) with the ontology being debugged which is called $\mathcal{O}_{debug}$ in the rest of the paper. At the end of the first step, we have a merged ontology that is used in the next step to detect MUPS sets. In the third step, all of the MUPS sets are analyzed altogether to detect the incorrect axioms. Repairing the incorrect axioms is the last step. We discuss the details of these steps in the following subsections.

Assumption 1. *There is no error in the profile ontology.*

Here, we make Assumption 1 about the correctness of the axioms in the profile ontology. It means that we trust in the knowledge of the profile ontology. In this manner, the profile ontology is a correct representation of the real world. Selecting ontologies as contextual knowledge could be done by human experts that perform ontology debugging.

Researches in the ontology evaluation field suggest criteria to ensure validity and verification of the profile ontology. Vrandečić (2009) discusses some quality criteria for ontology verification. Obrst et al. (2007) suggests requirements for ontology validation. Neuhaus et al. (2014) reports existing best practices and tools in the field of ontology evaluation. As discussed in Gelernter & Jha (2016); Vrandečić (2009), choosing the ontology evaluation criteria that fits into an application mostly needs a human level intelligence. Hence, the selection of ontologies as the profile ontology needs human expertise.

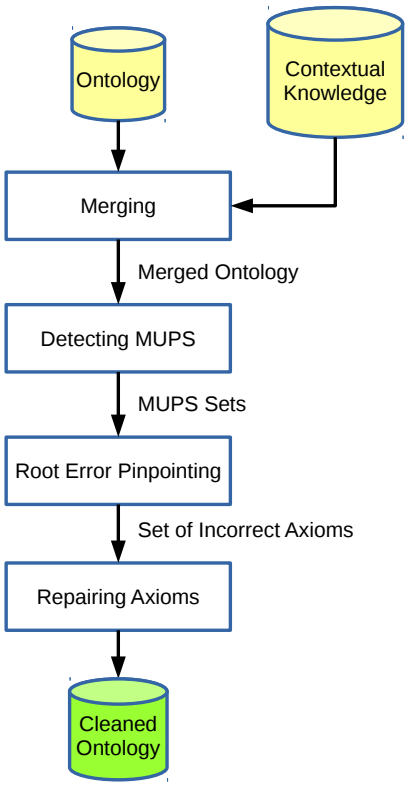

Figure 2: The Process of Ontology Debugging using Contextual Knowledge

One of the most important selection criteria is the relatedness of the profile ontology to the ontology being debugged in terms of the realm of the interest they are modeling. Having more logical axioms could be another criteria since, having more axioms could be a sign of better coverage of the knowledge in the real world.

One important thing about our methodology is that we only debug the TBox of an ontology to detect MUPS sets. Hence, we drop the ABox axioms before starting the process.

3.1. Step 1: Merging

The first step in using the knowledge of the profile ontology is to align the concepts and the relations in the signatures of $\mathcal{O}_{debug}$ and $\mathcal{O}_{profile}$. The alignment is a set of axioms that defines correspondences between concepts and relations of two ontologies and is defined formally via Definition 6.

Definition 6 (Alignment)**.**

$$\begin{aligned}\mathcal{A} \subseteq &\{xyz | x \in C_{\mathcal{O}_{debug}}, y \in \{\equiv, \sqsubseteq, \sqsupseteq\}, z \in C_{\mathcal{O}_{profile}}\} \\ \cup &\{xyz | x \in R_{\mathcal{O}_{debug}}, y \in \{\equiv, \sqsubseteq, \sqsupseteq\}, z \in R_{\mathcal{O}_{profile}}\}\end{aligned}$$

If the signature of $\mathcal{O}_{debug}$ is a subset of the signature of $\mathcal{O}_{profile}$ (*i.e.* $C_{\mathcal{O}_{debug}} \subseteq C_{\mathcal{O}_{profile}}$ and $R_{\mathcal{O}_{debug}} \subseteq R_{\mathcal{O}_{profile}}$), then there is no need to generate the alignment $\mathcal{A}$. In that case, $\mathcal{A}$ is the empty set ($\emptyset$).

After generating the alignment using a state-of-the-art ontology matcher, merging $\mathcal{O}_{debug}$ and $\mathcal{O}_{profile}$ will be easy as the union of their axioms. The merged ontology ($\mathcal{O}_m$) is defined in Definition 7.

Definition 7 (Merged Ontology)**.**

$$\mathcal{O}_m = \mathcal{O}_{debug} \cup \mathcal{O}_{profile} \cup \mathcal{A}$$

3.2. Step 2: Detecting MUPS

The story of the debugging process starts with unsatisfiable concepts. Ontology reasoners (*e.g.* Hermit, Pellet, ...) can be easily used to detect the list of unsatisfiable concepts in the ontology. Hence, we want to detect MUPSs for each unsatisfiable concept according to Definition 3.

The set of MUPS in $\mathcal{M}(\mathcal{O}_m)$ can be categorized into three types:

- **TYPE1:** All of its axioms belong to $\mathcal{O}_{debug}$ and formally defined as

  $$\mathcal{M}_{TYPE1} \in \{\mathcal{M} | \forall \alpha \in \mathcal{M} : \alpha \in \mathcal{O}_{debug}\}.$$

  These MUPSs are contained in $\mathcal{M}(\mathcal{O}_{debug})$, so they could also be detected using other debugging methods.

- **TYPE2:** Some of the axioms of the MUPS belong to $\mathcal{O}_{debug}$ and some of them belong to the profile ontology. TYPE2 MUPS is formally defined as

  $$\mathcal{M}_{TYPE2} \in \{\mathcal{M} | \exists \alpha \in \mathcal{M} : \alpha \in \mathcal{O}_{debug} \wedge \exists \beta \in \mathcal{M} : \beta \in \mathcal{O}_m \cup \mathcal{A}\}.$$

  These MUPS sets could be helpful in the detection of the hidden errors.

- **TYPE3:** All of the axioms in this type belong to the $\mathcal{O}_{profile}$ and are defined as

  $$\mathcal{M}_{TYPE3} \in \{\mathcal{M} | \forall \alpha \in \mathcal{M} : \alpha \in \mathcal{O}_m\}.$$

  These MUPSs show some incoherency in the background knowledge.

Figure 3 shows an example of MUPS types in terms of the introduced types. $\mathcal{M}_1$ is completely inside of $\mathcal{O}_{debug}$ and is categorized as TYPE1. $\mathcal{M}_2$ shows a TYPE2 MUPS, and finally, $\mathcal{M}_3$ is a TYPE3 MUPS which is completely inside the profile ontology.

TYPE3 shows some conflicts inside the profile ontology. The desirable results will be achieved if there is no TYPE3 MUPS. To this end, the one who uses this approach is responsible to choose some correct background knowledge.

Since we are debugging $\mathcal{O}_{debug}$, we are not interested in the third type, and they are not processed in this step. Thus, the main focus will be on MUPS sets of TYPE1 and TYPE2.

MUPSs belonging to TYPE2 reveal some conflicts between some axioms from the profile ontology and some axioms from $\mathcal{O}_{debug}$. In order to define hidden errors in $\mathcal{O}_{debug}$, the knowledge of the ontology is evaluated with respect to the profile ontology. In this way we could define the Suspected Hidden Error as Definition 8.

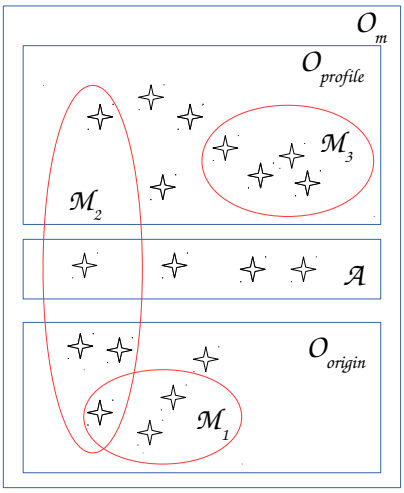

Figure 3: Example of MUPS sets positioning in $\mathcal{O}_m$. Axioms are shown by stars and ellipses represent MUPS sets. $\mathcal{M}_1$ is TYPE1, $\mathcal{M}_2$ is TYPE2 and $\mathcal{M}_3$ is TYPE3 MUPS.

Definition 8 (Suspected Hidden Error)**.** *A suspected hidden error axiom is an axiom from $\mathcal{O}_{debug}$ that takes part only in TYPE2 MUPSs. This could be formally defined as:*

α is a suspected hidden error iff:

$$\alpha \in \mathcal{O}_{debug}, \forall \mathcal{M} : \alpha \in \mathcal{M} \rightarrow \mathcal{M} \in \mathcal{M}_{TYPE2}(\mathcal{O}_m)$$

where $\mathcal{M}_{TYPE2}(\mathcal{O}_m)$ is the set of all TYPE2 MUPS in $\mathcal{M}(\mathcal{O}_m)$.

Reiter (1987) proofed that the set of minimal diagnosis for an unsatisfiable concept is equivalent to the minimal hitting set of MUPS of the concept. Based on this theorem he proposed the HST (Hit Set Tree) algorithm (Reiter, 1987) to enumerate all the MUPS of an unsatisfiable concept. HST is a sound and complete algorithm (see Kalyanpur et al. (2007); Reiter (1987) for the proof of it). Algorithm 1 shows the pseudo-code of the hit set algorithm.

The HST algorithm constructs a search tree, and each node of the search tree is an MUPS or a diagnosis. The MUPS nodes are expanded in the breadth-first mode to build the hit set tree. A first-in-first-out (FIFO) queue is used to maintain the list of not expanded nodes. The algorithm starts with a random MUPS as the root node of the tree.

Each MUPS node is expanded, and, for each axiom inside it, a new child node is generated. The edge is labeled with the associated axiom. The algorithm examines the new path from the root to the new node. The algorithm first checks for early termination conditions (which will be discussed below). If the early termination conditions are satisfied, then the new branch will be marked as $\square$ and the expansion of the node will be stopped. If not, the algorithm calls the random MUPS detector procedure to check if there is an MUPS inside the ontology resulting from removing the axioms of the new path from the debugging ontology. If it could detect an MUPS in the absence of

Algorithm 1 HST Algorithm for Enumerating All MUPS of an Unsatisfiable Concept.

Require: O, Unsatisfiable Concept C

Ensure: $\mathcal{M}_C$ is set of MUPSs which preserve unsatisfiability of Concept C

procedure FINDMUPSS(O, C)
  $\mathcal{M}_C \leftarrow \emptyset$, $\mathcal{D}_C \leftarrow \emptyset$
  $\mathcal{M} \leftarrow$ GENERATERANDOMMUPS(O, C)
  $ENQUEUE(notExpandedNodes, < \mathcal{M}, \emptyset >$
  while $notExpandedNodes$ is not empty **do**
    $< \mathcal{M}, path > \leftarrow DEQUEUE(notExpandedNodes)$
    for all $\alpha \in \mathcal{M}$ **do**
      $newPath \leftarrow path \cup \{\alpha\}$
      if $\nexists \mathcal{D}' \in \mathcal{D}_C \; s.t. \; \mathcal{D}' \subseteq newPath$ and $\nexists < \mathcal{M}', path' > \in notExpandedNodes \; s.t. \; path' = newPath$ **then** ▷ path is not examined before
        $\mathcal{M}' \leftarrow$ GENERATERANDOMMUPS($O \setminus newPath, C$)
        if $\mathcal{M}' = \emptyset$ **then** ▷ newPath is a diagnosis!
          $\mathcal{D}_C \leftarrow \mathcal{D}_C \cup newPath$
        else ▷ new MUPS is found
          $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \mathcal{M}'$
          $ENQUEUE(notExpandedNodes, < \mathcal{M}', newPath >$
        end if
      end if
    end for
  end while
end procedure

$\mathcal{O}_{debug}$	α_1 :	$WaterwayTunnel \sqsubseteq Place$
	α_2 :	$Infrastructure \sqsubseteq \neg Place$
	α_3 :	$Place \sqsubseteq \neg WaterwayTunnel$
$\mathcal{O}_{profile}$	α_4 :	$Infrastructure \sqsubseteq ArchitecturalStructure$
	α_5 :	$ArchitecturalStructure \sqsubseteq Place$
	α_6 :	$RouteOfTransportation \sqsubseteq Infrastructure$
	α_7 :	$WaterwayTunnel \sqsubseteq RouteOfTransportation$

Figure 4: An Example of Merged Ontology Taken From a Real Debugging Task. In this example, concept $WaterwayTunnel$ is unsatisfiable.

the new path's axioms, then the new node with the detected MUPS and the new path will be added to the queue for further expansion. If the unsatisfiable concept becomes satisfiable in the absence of the new path's axiom, it means that the new path is a diagnosis for the concept. In this situation, the new node will be marked as ✓ and the new path will be added to the set of diagnoses. Hence, removing the axioms in the path from ontology yields the satisfiability of the concept. The queue processing for the expansion of the tree will be continued until the $notExpandedNodes$ become empty.

The algorithm is optimized by not expanding the paths that satisfy one of the early termination conditions. The early termination conditions are:

1. If the new path (set of the axioms in the path) is a superset of some diagnosis, then the new path is also a diagnosis and there is no need to be expanded more. This is due to the simple fact that any superset of a satisfiable path is also a satisfiable path.
2. If there is another not-expanded node with the same path, then the new path is a duplication of the waiting path and expanding new path will be redundant.

Algorithm 1 exploits a procedure named $GenerateRandomMUPS$ to detect a random MUPS for an unsatisfiable concept in a given ontology. Generally, it employs the expand-shrink strategy to get a random MUPS. The expand phase starts with an empty set of axioms and then a random number of axioms are included to get a set of axioms $M \subseteq O \; s.t. \; M \models C \sqsubseteq \bot$. Then, at the shrink phase, M will be reduced to get $\mathcal{M}$ which complies with the minimality of MUPS in Definition 3. See Kalyanpur (2006) for the details of $GenerateRandomMUPS$.

Figure 4 shows a merged ontology that consists of 3 axioms ($\{\alpha_1, \alpha_2, \alpha_3\}$) from $\mathcal{O}_{debug}$ and 4 axioms ($\{\alpha_4, \alpha_5, \alpha_6, \alpha_7\}$) from $\mathcal{O}_{profile}$. Let's assume that $\mathcal{O}_{debug}$ and $\mathcal{O}_{profile}$ have the same signature, so the alignment $\mathcal{A}$ is an empty set. In this ontology, concept $WaterwayTunnel$ is an unsatisfiable concept. The hit set tree built by Algorithm 1 is shown in Figure 5.

Let's assume that $GenerateRandomMUPS$ detects randomly $\mathcal{M}_1$ as the root node of the tree. The algorithm tries to expand the root node by making a branch for each axiom in $\mathcal{M}_1$. The tree is expanded using breadth-first strategy. Thus, the root node is expanded for α_1 and the edge is labeled α_1 (left branch). Since there is no diagnosis at the start of the algorithm and there is no awaiting node in the queue, the early termination condition is not satisfied. The algorithm tries to detect an MUPS in $O \setminus \{\alpha_1\}$. $GenerateRandomMUPS$ returns $\mathcal{M}_2$. So, the new

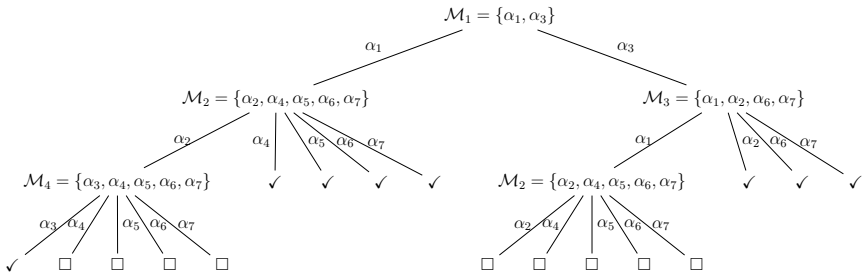

Figure 5: Hit Set Tree built by Algorithm 1 to enumerate all MUPS of the Concept $WaterwayTunnel$. Nodes of the tree represent MUPS sets. Paths marked by ✓ are diagnosis and those marked by □ are early terminated ones due to some early termination condition.

node is marked as MUPS node and $< \mathcal{M}_2, \{\alpha_1\} >$ is queued. In a similar way $< \mathcal{M}_3, \{\alpha_3\} >$ is enqueued. Expanding $< \mathcal{M}_2, \{\alpha_1\} >$ results in the new node $< \mathcal{M}_4, \{\alpha_1, \alpha_2\} >$. The other four nodes will be marked ✓ as satisfiable paths. The rest of tree is expanded in a similar way. The nodes with □ label are early terminated due to the first condition. For example, the path $[\alpha_1 - \alpha_2 - \alpha_4]$ is early terminated since it's a superset of $\{\alpha_1, \alpha_4\}$. The set of MUPS nodes of the tree are $\mathcal{M}(WaterwayTunnel) = \{\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4\}$.

Running Algorithm 1 for the unsatisfiable concept *Infrastructure* will result in finding $\mathcal{M}_5 = \{\alpha_2, \alpha_4, \alpha_5\}$ and running it for *RouteOfTransportation* will result in finding $\mathcal{M}_6 = \{\alpha_2, \alpha_4, \alpha_5, \alpha_6\}$.

3.3. Step 3: Root Error Pinpointing

Detecting MUPS sets in the previous step helps us to separate the free axioms from the suspected ones. The axioms that are part of some MUPS are suspected axioms. Some of these axioms are incorrect and are the root cause of incoherency in the ontology. Therefore the goal of this step is to find incorrect axioms in the MUPS sets. The set of these incorrect axioms is a diagnosis for the ontology. Before proceeding with root error detection, we make two assumptions to reduce the complexity of debugging.

Assumption 2. *All the axioms in the alignment $\mathcal{A}$ are correct.*

In Assumption 1, we trust in the axioms in the profile ontology. Assumption 2 is also about trusting in the axioms which align $\mathcal{O}_{debug}$ to $\mathcal{O}_{profile}$. If the alignment is manually revised (*i.e.* incorrect correspondences in $\mathcal{A}$ are manually removed by some human expert) or if it is a reference alignment, then Assumption 2 will be realistic.

According to Assumptions 1 and 2, all of the axioms in $\mathcal{O}_{profile}$ and $\mathcal{A}$ are correct. Hence the axioms of $\mathcal{O}_{profile}$ or $\mathcal{A}$ contained in the MUPS set $\mathcal{M}(\mathcal{O})$ are correct, only the axioms contained in $\mathcal{O}_{debug}$ could be responsible for the bugs detected.

Now we can define the candidate error set as Definition 9. The candidate error set E should be one of the diagnosis $\mathcal{D}(\mathcal{O}_m)$ such that all of it's axioms belong to $\mathcal{O}_{debug}$.

Definition 9 (Candidate Error Set)**.** *$E \in \mathcal{D}(\mathcal{O}_m)$ is an error set for $\mathcal{O}_m$ iff $E \subseteq \mathcal{O}_{debug}$.*

Algorithm 2 Greedy Algorithm to Detect the Error Set

Require: $\mathcal{M}(\mathcal{O}_m), Cost(\alpha)$
Ensure: $\mathcal{E}$ is an Error Set at the end of algorithm
1: $\mathcal{E} \leftarrow \emptyset$
2: $unCoveredMUPS \leftarrow$ all MUPS $\in \mathcal{M}(\mathcal{O}_m)$
3: **while** $unCoveredMUPS \neq \emptyset$ **do**
4: $axioms \leftarrow \{\alpha | \alpha \in mups \ : \ mups \in unCoveredMUPS\}$
5: sort $axioms$ ASC using $Cost(\alpha)$
6: $i \leftarrow 0$
7: **while** $i < length(axioms)$ && $axioms[i] \in \mathcal{O}_{profile} \cup \mathcal{A}$ **do**
8: $i \leftarrow i + 1$
9: **end while**
10: **if** $i! = length(axioms)$ **then**
11: $\alpha_{min} \leftarrow axioms[i]$
12: **else**
13: $\alpha_{min} \leftarrow axioms[0]$
14: **end if**
15: $\mathcal{E} \leftarrow \mathcal{E} \cup \{\alpha_{min}\}$
16: $unCoveredMUPS \leftarrow \{\mathcal{M} | \nexists \alpha \in \mathcal{M} : \alpha \in \mathcal{E}\}$
17: **end while**

The candidate error sets $E(\mathcal{O}_m)$ should be evaluated to select one of them as the *Error Set* such that removing them from ontology will result in a minimal cost. So we should be able to evaluate axioms of each candidate error set to select the one with the least effect on the ontology.

Definition 10 (Error Set)**.** *$\mathcal{E}$ is a candidate error set iff it minimize the* cost function *over it's axioms.*

$$\mathcal{E} \in \arg\min_{E} \sum_{\alpha \in E} Cost(\alpha)$$

Definition 10 defines detecting root causes of incoherency as an optimization problem. Since finding all the candidate error sets and choosing the error set with the minimum cost is not a polynomial time algorithm, we use Algorithm 2 which exploits a greedy approach to construct the error set. The complexity of the algorithm is $O(n)$

Algorithm 2 starts with an empty $\mathcal{E}$ and select the axiom α_{min} with the minimum cost which is not contained in $\mathcal{O}_{profile} \cup \mathcal{A}$ as an error and adds it to $\mathcal{E}$. In TYPE 3 MUPS sets (which all of its axioms are contained in $\mathcal{O}_{profile} \cup \mathcal{A}$), the algorithm cannot find an axiom from $\mathcal{O}_{debug}$ to cover it (in lines 7-9), So it chooses the first axiom with minimum cost as an error (line 13). It will continue to repeat this loop and adds more such axioms until $\mathcal{E}$ turns into a hit set for $\mathcal{M}(\mathcal{O}_m)$ and could cover all the MUPSs in $\mathcal{M}(\mathcal{O}_m)$. At this point, it will terminate and ensure that $\mathcal{E}$ is a hitting set for $\mathcal{M}(\mathcal{O}_m)$ with the minimum cost.

Defining the $Cost$ function has a major role in detecting the error set. This function is used to rank the axioms when looking for the most guilty axiom in MUPS. In fact, it defines to what extent an axiom is incorrect with respect to the others. When an axiom

has a lower cost value, it indicates that removing this axiom from ontology will not result in much knowledge loss. So the cost function should return the lowest cost value for the incorrect axioms. In the following subsections, we introduce three new cost functions. Also, the cost function used in the SWOOP ontology debugging tool (Parsia et al., 2005), is introduced to be used as the baseline in the evaluation of our new cost functions.

3.3.1. Support Cost Function

Since the knowledge in profile (background context ontologies) are trusted, Support cost function uses it as a reference knowledge in evaluating axioms. Definition 11 defines Support value for an axiom as a sum of support value each ontology in the profile provides with the axiom. If an axiom is entailed by an ontology in the knowledge-base profile, it is supposed that the axiom has support from that ontology. But if the negation of the axiom is entailed by the ontology, then the ontology is against the axiom and have a negative support for that axiom.

Definition 11 (Support Function)**.** *Support Function determines how much support for an axiom exists in the background knowledge*

$$Supp(\alpha) = \sum_{\mathcal{O}_i \in \mathcal{PROFILE}} Supp_{\mathcal{O}_i}(\alpha)$$

where,

$$Supp_{\mathcal{O}}(\alpha) = \begin{cases} K & \text{if } \mathcal{O} \models \alpha \\ -K & \text{if } \mathcal{O} \models \neg\alpha \\ 0 & \text{otherwise} \end{cases}$$

*K is a large positive number (*e.g. $K = 1000$*).*

Example 1. *In the ontology $\mathcal{O}_{debug}$ as shown in the Figure 4, $\mathcal{O}_{profile} \models \alpha_1$. So $Supp_{\mathcal{O}_{profile}}(\alpha_1) = K$. In the same ontology, the negation of α_3 is entailed from $\mathcal{O}_{profile}$ ($\mathcal{O}_{profile} \models \neg\alpha_3$), so the support value of profile ontology for α_3 will be $-K$. It means that cost of removing α_3 is much lower than α_1 according to Support function. Hence, using Support function in Algorithm 2 will result in choosing α_3 as error axiom.*

3.3.2. ShapleyMI Cost Function

Hunter & Konieczny (2010) used the idea of Shapley Value from the game theory to compute some penalty value for the axioms which take part in some inconsistency sets. Shapley Minimum Inconsistency Value (S^{MI}) is defined as Definition 12.

Definition 12 (Shapley Minimum Inconsistency Value)**.**

$$S^{MI}(\alpha) = \sum_{\mathcal{M} \in \mathcal{M}(\mathcal{O}_m) \text{ s.t. } \alpha \in \mathcal{M}} \frac{1}{|\mathcal{M}|}$$

S^{MI} assumes that each axiom of $\mathcal{M}$ has equal role in bringing $\mathcal{M}$ into existence. Hence, S^{MI} assigns the penalty of α in MUPS $\mathcal{M}$ inversely proportional to the size of

$\mathcal{M}$. In fact, $\frac{1}{|\mathcal{M}|}$ is the role of each axiom in the existence of such conflict set. If the axiom is in more than one MUPS, the total penalty will be sum of all such penalties.

ShapleyMI Cost Function (as defined by Definition 13) assumes that the probability of being an error axiom is directly proportional to the penalty value (S^{MI}) an axiom has. Since our algorithm assumes that error axioms have less cost, we defined ShapleyMI function equal to the inverse of S^{MI}.

Definition 13 (ShapleyMI Function)**.**

$$ShapleyMI(\alpha) = \frac{1}{S^{MI}(\alpha)}$$

Example 2. *α_4 in the ontology shown in Figure 4, takes part in MUPS sets $\mathcal{M}_2 = \{\alpha_2, \alpha_4, \alpha_5, \alpha_6, \alpha_7\}$ and $\mathcal{M}_4 = \{\alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7\}$. So the axiom α_4 will get penalty value as $S^{MI}(\alpha_4) = \frac{1}{5} + \frac{1}{5} = 0.4$ and ShapleyMI function computes the cost of removing this axiom as $\frac{1}{0.4} = 2.5$.*

Example 3. *In the same ontology, α_3 takes part in $\mathcal{M}_4$ and $\mathcal{M}_1 = \{\alpha_1, \alpha_3\}$. So the penalty value of α_3 will be $S^{MI}(\alpha_3) = \frac{1}{5} + \frac{1}{2} = 0.7$. The ShapleyMI value of α_3 is $\frac{1}{0.7} \approx 1.43$ that is smaller than the ShapleyMI value of α_4. It means that removing α_3 has less cost than removing α_4. In other words, since α_3 has more penalty than α_4, our algorithm prefers to choose α_3 as error axiom in compare to α_4.*

3.3.3. ShapleySupport Cost Function

Support cost function has a weakness that it gives the same cost value either for all of the axioms that are supported by the profile or for all of the axioms that the negation of them are supported (*i.e.* either $+K$ or $-K$). So this function does not distinguish between each of the supported axioms nor not supported axioms.

At the other side, ShapleyMI cost function evaluates axioms by considering the frequency of appearance of axioms in MUPS sets as well as the size of such MUPS sets. ShapleyMI considers the positioning of the axioms in the MUPS sets. So this function seems to be a good choice to cover the weakness of the Support function which exploits only the semantic of axioms.

We define ShapleySupport cost function as Definition 14. It combines the *Shapley Value* into *Support Value*.

Definition 14 (ShapleySupport Function)**.**

$$ShapleySupport(\alpha) = \begin{cases} Supp(\alpha) * ShapleyMI(\alpha) & \text{if } Supp(\alpha) > 0 \\ ShapleyMI(\alpha) & \text{if } Supp(\alpha) = 0 \\ Supp(\alpha)/ShapleyMI(\alpha) & \text{if } Supp(\alpha) < 0 \end{cases}$$

Example 4. *In the example ontology of Figure 4, the negation of α_2 and α_3 is supported by $\mathcal{O}_{profile}$. The cost value of Support function for these axioms is $-K$. In this way, Support function could not distinguish between α_2 and α_3 and the degree of being error is the same for both axioms. Computing ShapleySupport cost value for these axioms results in $ShapleySupport(\alpha_2) = \frac{-1000}{0.97} = -2066.67$ and $ShapleySupport(\alpha_3) = \frac{-1000}{1.43} = -1400$. In this way, ShapleySupport differentiate between two unsupported axioms and assigns α_2 less cost value than α_3.*

3.3.4. SWOOP Cost Function

Swoop ontology editing and debugging tool (Kalyanpur et al., 2006) is one of the systems which perform well in debugging OWL ontologies (Stuckenschmidt, 2008). So in this section, we introduce the method used in Swoop to rank the suspected axioms. The Swoop method used as a baseline for evaluating the methods proposed in the previous subsections.

Kalyanpur (2006) used three factors to rank the suspected axioms as an error and implemented them in the Swoop editor. These three factors are:

Frequency : Number of MUPS that the axiom participates in them.

Semantic Impact : Number of entailments that will be lost if the axiom is removed from the ontology.

Usage : This factor is about in what extent the signature of the axioms is used in the signature of the other axioms and is equal to the percent of axioms that include the signature of the axiom.

Swoop calculates the weighted sum of these factors to obtain the value of each axiom which is used as cost value. The definition of the Swoop cost function is as Definition 15.

Definition 15 (Swoop Function)**.**

$$swoop(\alpha) = w_0/Freq(\alpha) + w_1 * Impact(\alpha)/max_{Impact} + w_2 * Usage(\alpha)$$

where max_{Impact} is the maximum impact among all suspected axioms that is used to normalize the impact value between [0,1].

Example 5. *Calculation of $swoop(\alpha_2)$ is as:*

- $Freq(\alpha_2) = 4$*, as α_2 is participating in $\mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_5$ and $\mathcal{M}_6$.*
- $Impact(\alpha_2) = 1$*,*
- $Usage(\alpha_2) = \frac{6}{7} = 0.86$
- $max_{Impact} = 10$ *(semantic impact of α_6 and α_7 is 10)*

Using 0.9, 0.7 and 0.1 respectively for w_1, w_2 and w_3 results in $Swoop(\alpha_2) = 0.9 * 4 + 0.7 * 1/10 + 0.1 * 0.86 = 3.76$

3.4. Step 4: Repairing Axioms

Treating axioms detected as incorrect ones is the last step in debugging. The simplest solution is to eliminate them from the ontology. Since the error set ($\mathcal{E}$) is a hitting set for the set of MUPS ($\mathcal{M}(\mathcal{O}_m)$), removing them will result in an ontology which has no unsatisfiable concepts. Thus, the ontology will be coherent. The alternative option is to repair axioms automatically or manually using a human ontology expert. Since this paper is about detecting errors, not repairing axioms, we leave it to the user to decide what kind of repair is appropriate for each incorrect axiom.

4. Evaluations

We implemented our proposed method as a configurable Java tool. OntoDebugger[2], the implemented tool, uses OWL API[3] v4.x library for managing ontologies. It can use any standard reasoner that supports OWL API (*e.g.* Pellet, Hermit, ...) to check satisfiability of concepts. As described before, the standard reasoner is used as a black box for checking satisfiability of concepts. However, all the debugging functions are implemented inside OntoDebugger. Hermit[4] is used as the standard ontology reasoner in our experiments.

To evaluate the effect of the proposed method on detecting hidden errors in ontologies, we did our experiments on two different ontology sets. In the following subsections, we describe our test cases. In each case, we analyze the effect of adding background knowledge in detecting hidden errors in terms of the number of unsatisfiable concepts and the number of TYPE2 MUPS sets.

4.1. Case 1: Debugging Automatically Learned Disjointness Ontology

The ontology used in this experiment is learned automatically by analyzing statistical schema induction on DBpedia instances (see Fleischhacker et al. (2012); Völker & Niepert (2011)) and generated by GoldMiner (Fleischhacker & Völker, 2011) tool. The learned axioms mostly include *Disjointness* axioms between concepts of DBpedia ontology and have $\mathcal{ALCH}$ expressiveness. Völker et al. (2015) published a high-quality gold standard for class disjointness of DBpedia concepts.

Fleischhacker et al. (2013) used this ontology to construct a dataset of 11 ontologies named $O_0, O_1, \cdots, O_{10}$. Constructing the dataset started with randomly choosing 20% of the axioms in the ontology as O_0 and gradually adding some random axioms to the previous ontology to get a new ontology. Hence, ontologies with larger indices contain ontologies with lower indices (*i.e.* $O_i \subseteq O_{i+1}$). These ontologies contain only T-Box axioms. The summary of the dataset is given in Table 1. Based on the gold standard, approximately 1% of axioms of each ontology are erroneous axioms.

The dataset constructed by Fleischhacker et al. (2013) allows us to answer the questions like "Is there any relation between the size of the knowledge base and the number of hidden errors in the ontology?" or "Is finding errors in small ontologies easier than large ontologies?". Also, we can analyze the performance of the debugging methods while debugging ontologies with larger axiom sets.

Since the dataset contains ontologies that enrich the ontology of DBpedia, we used DBpedia itself as the profile ontology to evaluate the correctness of the newly learned axioms. We used DBpedia ontology 2015-04 which has 26697 axioms with $\mathcal{ALCHF}(\mathcal{D})$ expressiveness. Some of the axioms in DBpedia ontology are also in our dataset ontologies (*i.e.* $DBpedia \cap O_i \neq \emptyset$). Since the dataset enriched the DBpedia ontology, it has the same signature as DBpedia ontology. Therefore, there is no need to use alignment between them. Hence, the alignment $\mathcal{A}$ in this experiment is the empty set ($\emptyset$).

[2]OntoDebugger will be published publicly soon.
[3]`https://github.com/owlcs/owlapi`
[4]`http://www.hermit-reasoner.com/`

Table 1: Summary of information about ontologies in the dataset learned from DBpedia

Ont.	# Axioms	# Classes	# Prop.	# Error Axioms
O_0	23,702	301	667	225
O_1	32,821	305	684	306
O_2	41,942	314	700	398
O_3	51,030	314	716	481
O_4	60,122	317	726	570
O_5	69,224	321	730	654
O_6	78,335	323	733	731
O_7	87,426	324	738	820
O_8	96,521	324	740	901
O_9	105,628	324	741	988
O_{10}	114,731	324	742	1,084

Table 2: Results of MUPS detection at step 2 are summarized. The number of the detected MUPS is categorized into three types introduced in Section 3.2.

Ontology	# TYPE1	# TYPE2	# TYPE3
O_{0m}	6	244	3
O_{1m}	13	347	3
O_{2m}	21	585	3
O_{3m}	32	705	3
O_{4m}	46	755	3
O_{5m}	73	998	3
O_{6m}	87	1084	3
O_{7m}	153	1328	3
O_{8m}	242	1465	3
O_{9m}	532	2102	3
O_{10m}	305	1695	3

As proposed in Section 3, we start the debugging with the merging step where we obtained a merged ontology by getting the union of axioms in DBpedia ontology and $O_i, i = 0, 1, \ldots, 10$. So we can obtain O_{im} as:

$$O_{im} = DBpedia \cup O_i \tag{5}$$

For the evaluation of our proposed idea, we start by looking at the number of MUPSs found in the debugging process. Table 2 shows how many MUPS are detected in O_{im}. Also, the categorization of MUPSs according to the types introduced in Section 3.2 (*i.e.* TYPE1, TYPE2, TYPE3) is shown. The second column (# TYPE1) shows how many MUPS are in the original ontology. As it can be seen in the third column (# TYPE2), lots of MUPSs are detected after adding the profile ontology in the debugging process. Since there are some incoherences in the DBpedia ontology 2015-04, 3 MUPS of TYPE3 are detected in the debugging process (the last column).

Table 3 shows how effective is the addition of DBpedia ontology as a profile ontology in the debugging process. The third column shows the number of error axioms that are part of some MUPSs when debugging only O_i, and the fourth column shows

Table 3: Hidden Errors Found in Debugging the Dataset using DBpedia as the Profile Ontology.

Ontology	# Total Errors	# Errors Participated in $\mathcal{M}(O_i)$	Errors Participated in $\mathcal{M}(O_{im})$	# Discovered Hidden Errors
O_0	225	3	20	17
O_1	306	6	25	19
O_2	398	10	31	21
O_3	481	17	40	23
O_4	570	26	47	21
O_5	654	35	56	21
O_6	731	41	64	23
O_7	820	52	74	22
O_8	901	66	82	16
O_9	988	83	92	9
O_{10}	1084	102	102	0

the number of errors that could be found when the profile ontology is added to the debugged ontology. The difference between these two columns (fifth column) shows how many hidden errors could be found when debugging ontology O_i with the profile ontology.

Although the number of error axioms in $\mathcal{M}(O_{im})$ is greater than $\mathcal{M}(O_i)$ (except for $i = 10$), the number of hidden errors found by our proposed method is decreasing as the size of the debugged ontology (O_i) is growing.

The main point here is that the ontology with fewer axioms has less knowledge to contradict with error axioms than the one with larger axioms count. For example, if there is an error axiom in the ontology with 10 axioms and another error axiom in another ontology with 1000 axioms, the probability of finding a contradiction in the ontology with 10 axioms is less than finding a contradiction in the ontology with 1000 axioms. In other words, errors could not be easily found in smaller ontologies.

In the case of O_{10}, contrary the other ontologies, adding DBpedia has no effect on finding hidden errors. As can be seen in the last row of Table 2, 1695 new (TYPE2) MUPSs are found in O_{10m}. But 0 discovered hidden errors in these 1695 MUPS shows that all TYPE2 MUPS are built using the same axioms which are participated in $\mathcal{M}(O_i)$.

4.2. Case 2: Debugging Ontologies in Web of Data

The ontologies debugged in this experiment are gathered from the web of data. Mostly, they are published in Linked Open Vocabularies (LOV[5]). About 495 ontologies were downloaded from LOV in Sep. 2015. Also, 238 other ontologies were searched using Google's search engine and added to LOV ontologies to make our dataset richer. A total of 733 ontologies are gathered.

[5] `https://lov.okfn.org/dataset/lov/`

Table 4: Summary of the Dataset used in the second debugging experiment.

Number of ontologies downloaded from LOV	495
Number of other ontologies downloaded from web	238
Number of ontologies skipped due to errors in loading/matching	126
Number of ontologies actually debugged	607
Number of incoherent ontologies in the dataset	16
Number of incoherent ontologies after merging with profile ontology	37
Number of discovered ontologies with hidden errors	**21**

We used DBpedia ontology 2015-04, which we have used in the previous experiment, as a profile ontology. For aligning the profile ontology and the debugged ontologies, we used AgreementMakerLight[6] (AML) ontology matcher. AML is one of the best ontology matchers according to the result of the competitions held by Ontology Alignment Evaluation Initiative[7] (OAEI). AML is an open source project which enabled us to easily integrate it with our implemented system. The alignments generated by AML were used after applying 0.8 as alignment acceptance threshold.

In the merging step, generating alignments for 21 ontologies were interrupted due to a 24-hour timeout. Also, loading ontologies in OWLAPI v4 or in Hermit reasoner encountered runtime exceptions for 105 other ontologies. Hence, these 126 ontologies are excluded from our initial dataset. Therefore, the number of debugged ontologies is reduced to 607.

The summary of the dataset and the result of adding DBpedia as a profile ontology in the debugging process are shown in Table 4. In our dataset, 21 ontologies are coherent, but after merging them with DBpedia, they become incoherent.

Table 5 shows the list of discovered erroneous ontologies. For each discovered ontology, the number of incoherent entities (concepts or object properties) is reported in the second column. The number of MUPS found in the ontologies is shown in the third column. The number of all suspected axioms in merged ontologies is shown in the fourth column, while the number of these suspected axioms that belong to the original ontologies is shown in the last column.

In fact, the last column shows the number of axioms that are "Free Axioms" in the original ontology, but our proposed method marks these axioms as "Suspected Axiom" after merging the original ontology with DBpedia.

4.3. Evaluation of Root Error Pinpointing

We've introduced four cost functions in Section 3.3 to evaluate suspected axioms and detect error axioms among them. In this section we compare the cost functions according to the precision, recall and f-measure of the error sets. Equations (6), (7) and

[6] `https://github.com/AgreementMakerLight`
[7] `http://oaei.ontologymatching.org/`

Table 5: Coherent Ontologies in the Dataset Which Become Incoherent After Merging With DBpedia.

Ontology	# Incoherent Entities	# MUPS	# Suspected Axioms	# Discovered Hidden Suspected Axioms
aiiso_2008-09-25	6	12	17	1
EDAM	36	259	94	70
ExtendedDnS	9	45	20	9
frapo_2014-01-31	6	12	17	1
frbr_2009-05-16	45	1276	169	18
ipo_2015-07-30	2	22	28	6
ka	6	13	22	3
km4c_2015-07-28	56	230	79	24
ma-ont_2013-03-20	24	103	108	10
omnfed_2015-04-04	76	174	94	4
ov_2011-11-25	2	2	14	2
schema_2015-08-06	154	235	196	142
sio_2015-04-21	29	32	40	9
sio	29	32	40	9
swc_2009-05-11	2	4	16	2
swrc_2007-06-22	6	13	22	3
tp_2015-06-08	1	1	6	1
umbel_2014-08-28	2	2	5	3
univ-bench	9	25	38	7
vag_2013-11-28	1	2	10	3
vivo-core-public-1.5	21	104	55	11

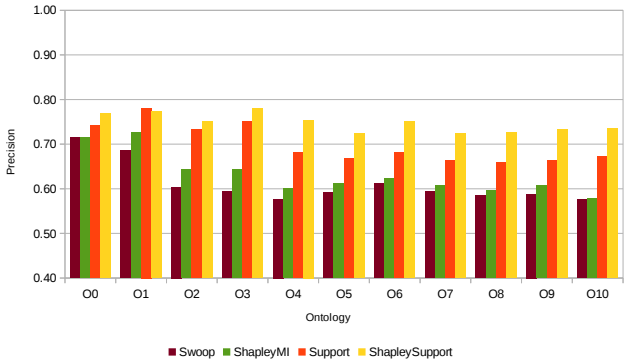

Figure 6: Comparison of Precision of the Cost Functions Introduced in Section 3.3

(8) define these measures.

$$precision = \frac{tp}{|\mathcal{E}|} \tag{6}$$

$$recall = \frac{tp}{\#\ truly\ errors\ in\ \mathcal{M}(O_m)} \tag{7}$$

$$f - measure = 2 * \frac{precision * recall}{precision + recall} \tag{8}$$

where tp is the number of real errors in the error set ($\mathcal{E}$).

The configurations of our experiments use 1000 as the value of the parameter K in the support-based functions. We used 0.9, 0.7 and 0.1 respectively for w_1, w_2 and w_3 of the Swoop method as used in the Swoop editor.

Figure 6 shows the precision of the cost functions used to detect error axioms in the ontology set learned from DBpedia (see Section 4.1). In all cases, the accuracy of the profile based functions (Support and ShapleySupport) is significantly better than Swoop function. Although ShapleyMI function is not significantly better than Swoop function, combining it with Support function (i.e. in ShapleySupport function) improves the accuracy of Support function.

As shown in Figure 7, the recall value of the cost functions are slightly equal and they could find an equal number of errors among suspected axiom sets. However, comparison of the f-measure values (as shown in Figure 8) shows the superiority of the support based methods.

According to the results of the experiments, evaluating the correctness of the axioms with respect to some trusted knowledge base (the profile ontology) significantly outperforms current methods which are mostly based on the frequency of participation of the axioms in the MUPS sets.

Comparing the run time of the cost functions (Figure 9) shows that ShapleyMI has the lowest runtime and Swoop function needs lots of time to compute axioms costs. Profile-based functions have much lower run time than Swoop method. Although Support and ShapleySupport need more time than ShapleyMI, the accuracy of these meth-

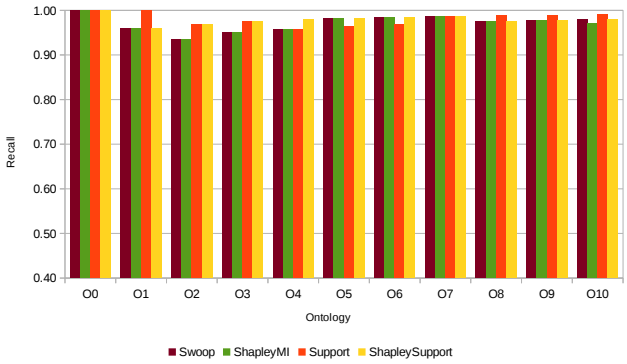

Figure 7: Comparison of Recall of the Cost Functions Introduced in Section 3.3

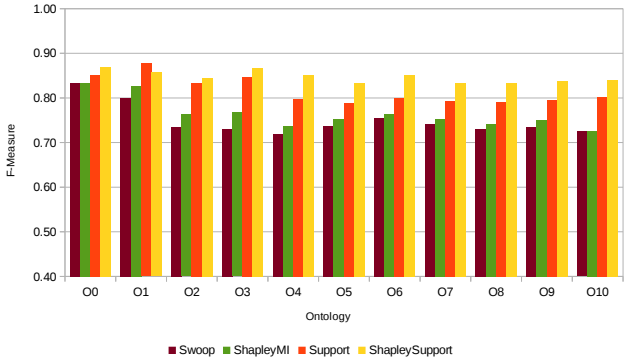

Figure 8: Comparison of F-Measure of the Cost Functions Introduced in Section 3.3

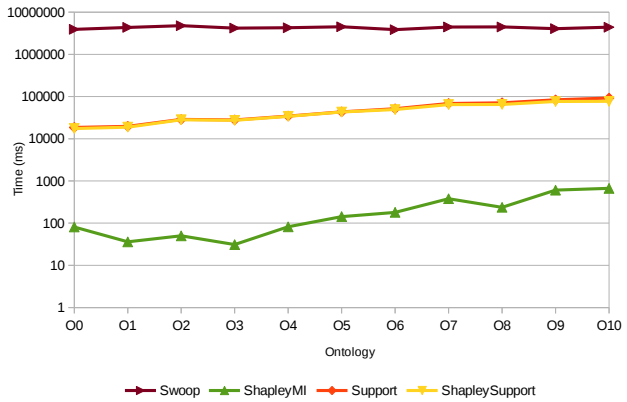

Figure 9: Run Time Comparison of the Cost Functions Introduced in Section 3.3

Table 6: Comparison of Three Error Pinpointing Functions in Detecting Errors in LOV Dataset.

Measure/Function	Swoop	ShapleyMI	ShapleySupport
True Positive	4	2	**5**
False Positive	**21**	24	29
Precision	**0.16**	0.08	0.15
Recall	0.09	0.05	**0.12**
F-Measure	0.12	0.06	**0.13**
Average Time	38 hours	**125 milliseconds**	2 minutes

ods justify preferring them to the ShapleyMI function.

Since we don't have ground truth for erroneous axioms of the ontologies in LOV dataset, we asked three ontology experts to evaluate axioms in MUPS sets discovered during Case 2 experiments (see Section 4.2). We used voting to accumulate the ontology experts evaluations. Then we compared the results of running root error pinpointing algorithms to the error set determined by ontology experts.

The set of suspected axioms for all of the 21 ontologies with hidden errors (see Table 5) has 789 unique axioms. Ontology experts marked 43 axioms as error axioms. The summary of evaluation results for the second dataset is shown in Table 6. From the previous experiments, it's obvious that Support and ShapleySupport functions have similar functionality. Since the results of ShapleySupport functions performed slightly better than the Support function, in this experiment we have omitted to evaluate Support function.

From Table 6, we can see that ShapleySupport has detected one more error axiom than the other methods. Swoop function detected less false error axioms but recall and f-measure of the ShapleySupport function are slightly better than the other methods. While the result of ShapleySupport does not outperform Swoop method, but having a reasonable run-time make it a better choice for detecting the root cause of errors in the ontologies.

5. Related Works

In this section, we discuss how our work is related to the other works in the field of ontology debugging. In this work, we used external knowledge to find logical errors in the terminology (T-Box) of the ontologies. In a similar way, Paulheim & Gangemi (2015) used DOLCE ontology as the background knowledge to find inconsistencies in the assertional axioms (A-Box) of DBpedia dataset. In Paulheim & Gangemi (2015) the axioms of DOLCE ontology are considered alongside DBpedia axioms. Using reasoners, they could find inconsistencies in the auto-generated axioms of DBpedia.

Our work is inspired by the doctoral thesis of Meilicke (2011), entitled "Alignment Incoherence in Ontology Matching". Meilicke targeted the debugging of ontology alignments in the field of ontology matching. He used two aligned ontologies as background knowledge to debug the generated alignment. If the generated alignment is considered as the ontology being debugged, the union of the two matched ontologies constructs the profile ontology to check incoherences in the generated alignment. We

generalized his methodology to offer a solution to debugging ontologies using other ontologies as the background knowledge.

In the literature, ontology debugging methods are categorized into two major approaches: Black-Box and White-Box methods. This categorization is based on how such methods use logical reasoning methods. Black-Box methods use standard reasoners as the oracle to ask questions regarding the coherency or consistency of ontologies. Our proposed method follows the black-box paradigm as well as other works like Schlobach et al. (2007); Kalyanpur et al. (2007); Horridge (2011); Baader & Suntisrivaraporn (2008); Suntisrivaraporn (2009); Ji et al. (2009).

White-box methods use special reasoning services which are designed specifically for ontology debugging. Extending tableau algorithms for checking coherency or consistency of ontologies is the basic technique used in the white-box methods. There are lots of white-box methods including Schlobach et al. (2007); Parsia et al. (2005); Kalyanpur et al. (2007); Baader & Peñaloza (2010). This categorization is not so rigid. Some methods including Schlobach et al. (2007); Kalyanpur et al. (2007) use both approaches in their methodologies.

There is another paradigm in the ontology debugging field, which relies on patterns in detecting ontology errors. These methods including Jarrar & Heymans (2008); Ji et al. (2012), provide some explanation for incoherent concepts that match some patterns in their pattern list. Although pattern-based methods are not considered as complete methods, they are more efficient compared to the other methods.

6. Conclusion

Previous works on ontology debugging find errors in the ontology by detecting logical contradictions (*e.g.* incoherences and inconsistencies). The assumption is that the erroneous axioms take part in some logical contradictions, and finding the contradictions could reduce the number of suspected axioms in finding the erroneous axioms. However, some error axioms might not contradict with other axioms in the ontology. Hence, detecting logical contradictions inside the ontology could not help finding those axioms. We call these *hidden errors* since there is no evidence for error regarding these axioms. We have proposed the idea of adding contextual background knowledge to the ontology in the ontology debugging process. Adding background knowledge might reveal some contradictions between hidden errors and the knowledge in the added ontology. Thus, hidden errors could be detected. Our results show that adding general background knowledge like DBpedia could result in discovering many hidden error axioms in the terminologies of the ontologies (T-Box). Many ontologies from LOV have been investigated and shown that logically coherent ontologies became incoherent after adding DBpedia as the background knowledge in the proposed ontology debugging process.

To the best knowledge of the authors, this is the first work that has suggested using background knowledge in the ontology debugging process, and we've shown that this could result in a remarkable increase in the number of discovered errors. The idea of this work is more useful when the ontology has a small number of axioms. In such ontologies, there might not be enough knowledge to contradict the error axioms, so error axioms could easily be hidden. Our results provide compelling evidence that the

search for errors in the ontology should not be limited to the logical contradictions inside the ontology. Checking the correctness of the knowledge in the ontology against other ontologies which model the same domain is necessary in order to detect hidden errors.

Selecting the appropriate background knowledge and matching it with the ontology is one of the challenges within the proposed process. The background knowledge should model the same domain as the ontology does. Although a match between background knowledge and the ontology could be generated using the state-of-art ontology matching methods, the correctness of the matching should be guaranteed.

Our proposed approach does not guarantee to find every hidden errors. It's a fact that no complete background knowledge exists in any domain. So, technically it could be some hidden errors that remain hidden even after adding some background knowledge.

Also in this work, we introduced three functions to evaluate MUPS axiom sets to pinpoint root causes of the contradictions in the ontologies. Suggested ShapleySupport function outperforms traditional methods like Swoop.

References

Arif, M. F., Mencía, C., Ignatiev, A., Manthey, N., Peñaloza, R., & Marques-Silva, J. (2016). BEACON: An Efficient SAT-Based Tool for Debugging $\mathcal{EL}^+$ Ontologies. In *Theory and Applications of Satisfiability Testing – SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings* (pp. 521–530). Cham: Springer International Publishing. doi:10.1007/978-3-319-40970-2_32.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (2010). *The Description Logic Handbook: Theory, Implementation and Applications*. (2nd ed.). New York, NY, USA: Cambridge University Press.

Baader, F., & Peñaloza, R. (2010). Axiom Pinpointing in General Tableaux. *Journal of Logic and Computation*, *20*, 5. doi:10.1093/logcom/exn058.

Baader, F., & Suntisrivaraporn, B. (2008). Debugging SNOMED CT Using Axiom Pinpointing in the Description Logic EL+. In *Proceedings of the Third International Conference on Knowledge Representation in Medicine, Phoenix, Arizona, USA, May 31st - June 2nd, 2008*. CEUR-WS.org volume 410 of *CEUR Workshop Proceedings*. URL: http://ceur-ws.org/Vol-410/Paper01.pdf.

Bell, D., Qi, G., & Liu, W. (2007). Approaches to Inconsistency Handling in Description-Logic Based Ontologies. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops* (pp. 1303–1311). Springer Berlin Heidelberg volume 4806 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-540-76890-6_58.

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - the story so far. *International Journal on Semantic Web and Information Systems*, *5*, 1–22. URL: http://eprints.soton.ac.uk/271285/.

Corcho, Ó., Roussey, C., Blázquez, L. M. V., & Pérez, I. (2009). Pattern-based OWL Ontology Debugging Guidelines. In *WOP*. CEUR-WS.org volume 516 of *CEUR Workshop Proceedings*.

Fleischhacker, D., Meilicke, C., Johanna, V., Niepert, M., & Völker, J. (2013). Computing Incoherence Explanations for Learned Ontologies. In *RR* (pp. 80–94). Springer volume 7994 of *Lecture Notes in Computer Science*.

Fleischhacker, D., & Völker, J. (2011). Inductive Learning of Disjointness Axioms. In *On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part II* (pp. 680–697). Springer volume 7045 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-642-25106-1_20.

Fleischhacker, D., Völker, J., & Stuckenschmidt, H. (2012). Mining RDF Data for Property Axioms. In *On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012,*

Rome, Italy, September 10-14, 2012. Proceedings, Part II (pp. 718–735). Springer. doi:10.1007/978-3-642-33615-7_18.

Friedrich, G. (2014). Interactive Debugging of Knowledge Bases. In *International Workshop on Principles of Diagnosis (DX'14)* (pp. 1–4).

Gelernter, J., & Jha, J. (2016). Challenges in Ontology Evaluation. *Journal of Data and Information Quality*, *7*, 1–4. doi:10.1145/2935751.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*, 199–220. doi:http://dx.doi.org/10.1006/knac.1993.1008.

Horridge, M. (2011). *Justification based explanation in ontologies*. Dissertation University of Manchester. URL: http://www.bcs.org/upload/pdf/dd-matthew-horridge.pdf.

Hunter, A., & Konieczny, S. (2010). On the measure of conflicts: Shapley Inconsistency Values. *Artificial Intelligence*, *174*, 1007–1026. doi:10.1016/j.artint.2010.06.001.

Jannach, D., Schmitz, T., & Shchekotykhin, K. (2016). Parallel Model-Based Diagnosis on Multi-Core Computers. *Journal of Artificial Intelligence Research*, *55*, 835–887.

Jarrar, M., & Heymans, S. (2008). Towards Pattern-Based Reasoning for Friendly Ontology Debugging. *International Journal on Artificial Intelligence Tools*, *17*, 607–634.

Ji, Q., Gao, Z., Huang, Z., & Zhu, M. (2012). An Efficient Approach to Debugging Ontologies Based on Patterns. In *The Semantic Web SE - 33* (pp. 425–433). Springer Berlin Heidelberg volume 7185 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-642-29923-0_33.

Ji, Q., Qi, G., & Haase, P. (2009). A Relevance-Directed Algorithm for Finding Justifications of DL Entailments. In *The Semantic Web: Fourth Asian Conference, ASWC 2009, Shanghai, China, December 6-9, 2009. Proceedings* (pp. 306–320). Springer Berlin Heidelberg. doi:10.1007/978-3-642-10871-6_21.

Kalyanpur, A., Parsia, B., Horridge, M., & Sirin, E. (2007). Finding All Justifications of OWL DL Entailments. In *ISWC/ASWC* (pp. 267–280). Springer volume 4825 of *Lecture Notes in Computer Science*.

Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., & Hendler, J. A. (2006). Swoop: A Web Ontology Editing Browser. *Journal of Web Semantics.*, *4*, 144–153.

Kalyanpur, A., Parsia, B., Sirin, E., & Hendler, J. A. (2005). Debugging unsatisfiable classes in OWL ontologies. *J. Web Sem.*, *3*, 268–293.

Kalyanpur, A. A. (2006). *Debugging and repair of owl ontologies*. Dissertation University of Maryland. URL: http://drum.lib.umd.edu/handle/1903/3820.

Lehmann, J., & Bühmann, L. (2010). ORE - A Tool for Repairing and Enriching Knowledge Bases. In *International Semantic Web Conference (2)* (pp. 177–193). Springer volume 6497 of *Lecture Notes in Computer Science*.

Meilicke, C. (2011). *Alignment Incoherence in Ontology Matching*. Dissertation University of Mannheim, Germany.

Meilicke, C., & Stuckenschmidt, H. (2008). Incoherence as a Basis for Measuring the Quality of Ontology Mappings. In *OM*. CEUR-WS.org volume 431 of *CEUR Workshop Proceedings*.

Moodley, K. (2010). *Debugging and repair of description logic ontologies*. Dissertation University of KwaZulu-Natal, Westville. URL: http://hdl.handle.net/10413/9762.

Neuhaus, F., Ray, S., & Sriram, R. D. (2014). Toward ontology evaluation across the life cycle. *Appl. Ontology*, *8*, 179–194. doi:10.6028/NIST.IR.8008.

Obrst, L., Ceusters, W., Mani, I., Ray, S., & Smith, B. (2007). The Evaluation of Ontologies. In *Semantic Web* (pp. 139–158). Springer US. doi:10.1007/978-0-387-48438-9_8.

Papacchini, F., & Schmidt, R. A. (2015). Debugging of ALC-Ontologies via Minimal Model Generation. In *Automated Reasoning Workshop 2015 Bridging the Gap between Theory and Practice ARW 2015* (pp. 5–6).

Parsia, B., Sirin, E., & Kalyanpur, A. (2005). Debugging OWL ontologies. In *WWW* (pp. 633–640). ACM. URL: http://dl.acm.org/citation.cfm?id=1060837.

Paulheim, H., & Gangemi, A. (2015). Serving DBpedia with DOLCE More than Just Adding a Cherry on Top. In *The Semantic Web - ISWC 2015 SE - 11* (pp. 180–196). Springer International Publishing volume 9366 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-319-25007-6_11.

Reiter, R. (1987). A Theory of Diagnosis from First Principles. *Artif. Intell.*, *32*, 57–95.

Rodler, P., Shchekotykhin, K., Fleiss, P., & Friedrich, G. (2013). RIO: Minimizing User Interaction in Ontology Debugging. In *Web Reasoning and Rule Systems: 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings* (pp. 153–167). Springer Berlin Heidelberg volume 7994 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-642-39666-3_12.

Roussey, C., & Zamazal, O. (2013). Antipattern detection: how to debug an ontology without a reasoner. In *WoDOOM* (pp. 45–56). CEUR-WS.org volume 999 of *CEUR Workshop Proceedings*.

Schlobach, S., Huang, Z., Cornet, R., & van Harmelen, F. (2007). Debugging Incoherent Terminologies. *J. Automated Reasoning*, *39*, 317–349.

Shchekotykhin, K. M., Rodler, P., Fleiss, P., & Friedrich, G. (2012). On Direct Debugging of Aligned Ontologies. In *International Semantic Web Conference (Posters & Demos)*. CEUR-WS.org volume 914 of *CEUR Workshop Proceedings*.

Stuckenschmidt, H. (2008). Debugging OWL Ontologies - A Reality Check. In *EON*. CEUR-WS.org volume 359 of *CEUR Workshop Proceedings*.

Stuckenschmidt, H. (2013). Debugging weighted ontologies. In *WoDOOM* (pp. 1–8). CEUR-WS.org volume 999 of *CEUR Workshop Proceedings*.

Suntisrivaraporn, B. (2009). *Polynomial-time reasoning support for design and maintenance of large-scale biomedical ontologies*. Ph.D. thesis Technischen Universita"t Dresden.

Völker, J., Fleischhacker, D., & Stuckenschmidt, H. (2015). Automatic acquisition of class disjointness. *Web Semantics: Science, Services and Agents on the World Wide Web*, *35, Part 2*, 124–139. doi:10.1016/j.websem.2015.07.001.

Völker, J., & Niepert, M. (2011). Statistical Schema Induction. In *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I* (pp. 124–138). Springer volume 6643 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-642-21034-1_9.

Vrandečić, D. (2009). Ontology Evaluation. In *Handbook on Ontologies* (pp. 293–313). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-92673-3_13.

Wang, P., & Xu, B. (2008). Debugging Ontology Mappings: A Static Approach. *Computing and Informatics*, *27*, 21–36. URL: http://www.cai.sk/ojs/index.php/cai/article/view/271/220.