Interactive ontology debugging: two query strategies for efficient fault localization[☆]

Kostyantyn Shchekotykhin[a,*], Gerhard Friedrich[a], Philipp Fleiss[a,1], Patrick Rodler[a,1]

[a]*Alpen-Adria Universität, Universitätsstrasse 65-67, 9020 Klagenfurt, Austria*

Abstract

Effective debugging of ontologies is an important prerequisite for their broad application, especially in areas that rely on everyday users to create and maintain knowledge bases, such as the Semantic Web. In such systems ontologies capture formalized vocabularies of terms shared by its users. However in many cases users have different local views of the domain, i.e. of the context in which a given term is used. Inappropriate usage of terms together with natural complications when formulating and understanding logical descriptions may result in faulty ontologies. Recent ontology debugging approaches use diagnosis methods to identify causes of the faults. In most debugging scenarios these methods return many alternative diagnoses, thus placing the burden of fault localization on the user. This paper demonstrates how the target diagnosis can be identified by performing a sequence of observations, that is, by querying an oracle about entailments of the target ontology. To identify the best query we propose two query selection strategies: a simple "split-in-half" strategy and an entropy-based strategy. The latter allows knowledge about typical user errors to be exploited to minimize the number of queries. Our evaluation showed that the entropy-based method significantly reduces the number of required queries compared to the "split-in-half" approach. We experimented with different probability distributions of user errors and different qualities of the a-priori probabilities. Our measurements demonstrated the superiority of entropy-based query selection even in cases where all fault probabilities are equal, i.e. where no information about typical user errors is available.

Keywords: Ontology Debugging, Query Selection, Model-based Diagnosis, Description Logic

1. Introduction

Ontology acquisition and maintenance are important prerequisites for the successful application of semantic systems in areas such as the Semantic Web. However, as state of the art ontology extraction methods cannot automatically acquire ontologies in a complete and error-free fashion, users of such systems must formulate and correct logical descriptions on their own. In most of the cases these users are domain experts who have little or no experience in expressing knowledge in representation languages like OWL 2 DL [2]. Studies in cognitive psychology, e.g. [3, 4], indicate that humans make systematic errors while formulating or interpreting logical descriptions, with the results presented in [5, 6] confirming that these observations also apply to ontology development. Moreover, the problem gets even more if an ontology is developed by a group of users, such as OBO Foundry[2] or NCI Thesaurus[3], is based on a set of imported third-party ontologies, etc. In this case inconsistencies might appear if some user does not understand or accept the *context* in which shared ontological descriptions are used. Therefore, identification of erroneous ontological definitions is a difficult and time-consuming task.

Several ontology debugging methods [7, 8, 9, 10] were proposed to simplify ontology development and maintenance. Usually the main aim of debugging is to

[☆]This article is a substantial extension of the preliminary results published in *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)* [1].

[*]Corresponding author at: Alpen-Adria Universität, Universitätsstrasse 65-67, 9020 Klagenfurt, Austria. Tel: +43 463 2700 3768, Fax:++43 463 2700 993768

Email addresses: kostya@ifit.uni-klu.ac.at (Kostyantyn Shchekotykhin), gerhard@ifit.uni-klu.ac.at (Gerhard Friedrich), pfleiss@ifit.uni-klu.ac.at (Philipp Fleiss), prodler@ifit.uni-klu.ac.at (Patrick Rodler)

[1]The research project is funded by grants of the Austrian Science Fund (Project V-Know, contract 19996)

[2]http://www.obofoundry.org

[3]http://ncit.nci.nih.gov

obtain a consistent and, optionally, coherent ontology. These basic requirements can be extended with additional ones, such as test cases [9], which must be fulfilled by the *target ontology* O_t. Any ontology that does not fulfill the requirements is *faulty* regardless of how it was created. For instance, an ontology might be created by an expert specializing descriptions of the imported ontologies (top-down) or by an inductive learning algorithm from a set of examples (bottom-up).

Note that even if all requirements are completely specified, many logically equivalent target ontologies might exist. They may differ in aspects such as the complexity of consistency checks, size or readability. However, selecting between logically equivalent theories based on such measures is out of the scope of this paper. Furthermore, although target ontologies may evolve as requirements change over time, we assume that the target ontology remains stable throughout a debugging session.

Given an set of requirements (e.g. formulated by a user) and a faulty ontology, the task of an ontology debugger is to identify the set of alternative diagnoses, where each diagnosis corresponds to a set of possibly faulty axioms. More concretely, a *diagnosis* $\mathcal{D}$ is a subset of an ontology O such that one should remove (change) all the axioms of a diagnosis from the ontology (i.e. $O \setminus \mathcal{D}$) in order to formulate an ontology O' that fulfills all the given requirements. Only if the set of requirements is complete the only possible ontology O' corresponds to the target ontology O_t. In the following we refer to the removal of a diagnosis from the ontology as a *trivial application* of a diagnosis. Moreover, in practical applications it might be inefficient to consider all possible diagnoses. Therefore, modern ontology debugging approaches focus on the computation of minimal diagnoses. A set of axioms $\mathcal{D}_i$ is a *minimal diagnosis* iff there is no proper subset $\mathcal{D}'_i \subset \mathcal{D}_i$ which is a diagnosis. Thus, minimal diagnoses constitute minimal required changes to the ontology.

Application of diagnosis methods can be problematic in the cases for which many alternative minimal diagnoses exist for a given set of test cases and requirements. A sample study of real-world incoherent ontologies, which were used in [8], showed that hundreds or even thousands of minimal diagnoses may exist. In the case of the Transportation ontology the diagnosis method was able to identify 1782 minimal diagnoses [4]. In such situations a simple visualization of all alternative sets of modifications to the ontology is ineffective. Thus an efficient debugging method should be able to discriminate between the diagnoses in order to select the *target diagnosis* $\mathcal{D}_t$. Trivial application of $\mathcal{D}_t$ to the ontology O allows a user to extend $(O \setminus \mathcal{D}_t)$ with a set of additional axioms EX and, thus, to formulate the target ontology O_t, i.e. $O_t = (O \setminus \mathcal{D}_t) \cup EX$.

One possible solution to the diagnosis discrimination problem would be to order the set of diagnoses by various preference criteria. For instance, Kalyanpur et al. [11] suggest a measure to rank the axioms of a diagnosis depending on their structure, usage in test cases, provenance, and impact in terms of entailments. Only the top ranking diagnoses are then presented to the user. Of course this set of diagnoses will contain the target diagnosis only in cases where the faulty ontology, the given requirements and test cases provide sufficient data to the appropriate heuristic. However, it is difficult to identify which information, e.g. test cases, is really required to identify the target diagnosis. That is, a user does not know a priori which and how many tests should be provided to the debugger to ensure that it will return the target diagnosis.

In this paper we present an approach for the acquisition of additional information by *generating* a sequence of queries, the answers of which can be used to reduce the set of diagnoses and ultimately identify the target diagnosis. These queries should be answered by an oracle such as a user or an information extraction system. In order to construct queries we exploit the property that different ontologies resulting from trivial applications of different diagnoses entail unequal sets of axioms. Consequently, we can differentiate between diagnoses by asking the oracle if the target ontology should entail a set of logical sentences or not. These entailed logical sentences can be generated by the classification and realization services provided in description logic reasoning systems [12, 13, 14]. In particular, the classification process computes a subsumption hierarchy (sometimes also called "inheritance hierarchy" of parents and children) for each concept description mentioned in a TBox. For each individual mentioned in an ABox, the realization computes all the concept names of which the individual is an instance [12].

We propose two methods for selecting the next query of the set of possible queries: The first method employs a greedy approach that selects queries which try to cut the number of diagnoses in half. The second method exploits the fact that some diagnoses are more likely than others because of typical user errors [5, 6]. Beliefs for an error to occur in a given part of a knowledge base, represented as a probability, can be used to estimate the change in entropy of the set of diagnoses if a

[4] In Section 5, we will give a detailed characterization of these ontologies.

particular query is answered. In our evaluation the fault probabilities of axioms are estimated by the type and number of the logical operators employed. For example, roughly speaking, the greater the number of logical operators and the more complex these operators are, the greater the fault probability of an axiom. For assigning prior fault probabilities to diagnoses we employ the fault probabilities of axioms. Of course other methods for guessing prior fault probabilities, e.g. based on context of concept descriptions, measures suggested in the previous work [11], etc., can be easily integrated in our framework. Given a set of diagnoses and their probabilities the method selects a query which minimizes the expected entropy of a set of diagnoses after an oracle answers a query, i.e. maximizes the information gain. An oracle should answer such queries until a diagnosis is identified whose probability is significantly higher than those of all other diagnoses. This diagnosis is most likely to be the target diagnosis.

In the first evaluation scenario we compare the performance of both methods in terms of the number of queries needed to identify the target diagnosis. The evaluation is performed using generated examples as well as real-world ontologies presented in Tables 8 and 12. In the first case we alter a consistent and coherent ontology with additional axioms to generate conflicts that result in a predefined number of diagnoses of a required length. Each faulty ontology is then analyzed by the debugging algorithm using entropy, greedy and "random" strategies, where the latter selects queries at random. The evaluation results show that in some cases the entropy-based approach is almost 60% better than the greedy one whereas both approaches clearly outperformed the random strategy.

In the second evaluation scenario we investigate the robustness of the entropy-based strategy with respect to variations in the prior fault probabilities. We analyze the performance of entropy-based and greedy strategies on real-world ontologies by simulating different types of prior fault probability distributions as well as the "quality" of these probabilities that might occur in practice. In particular, we identify the cases where all prior fault probabilities are (1) equal, (2) "moderately" varied or (3) "extremely" varied. Regarding the "quality" of the probabilities we investigate cases where the guesses based on the prior diagnosis probabilities are good, average or bad. The results show that the entropy method outperforms "split-in-half" in almost all of the cases, namely when the target diagnosis is located in the more likely two thirds of the minimal diagnoses. In some situations the entropy-based approach achieves even twice the performance of the greedy one. Only in cases where the initial guess of the prior probabilities is very vague (the bad case), *and* the number of queries needed to identify the target diagnosis is low, "split-in-half" may save on average one query. However, if the number of queries increases, the performance of the entropy-based query selection increases compared to the "split-in-half" strategy. We observed that if the number of queries is greater than 10, the entropy-based method is preferable even if the initial guess of the prior probabilities is bad. This is due to the effect that the initial bad guesses are improved by the Bayes-update of the diagnoses probabilities as well as an ability of the entropy-based method to stop in the cases when a probability of some diagnosis is above an acceptance threshold predefined by the user. Consequently, entropy-based query selection is robust enough to handle different prior fault probability distributions.

Additional experiments performed on big real-world ontologies demonstrate the scalability of the suggested approach. In our experiments we were able to identify the target diagnosis in an ontology with over 33000 axioms using entropy-based query selection in only 190 seconds using an average of five queries.

The remainder of the paper is organized as follows: Section 2 presents two introductory examples as well as the basic concepts. The details of the entropy-based query selection method are given in Section 3. Section 4 describes the implementation of the approach and is followed by evaluation results in Section 5. The paper concludes with an overview of related work.

2. Motivating examples and basic concepts

We begin by presenting the fundamentals of ontology diagnosis and then show how queries and answers can be generated and employed to differentiate between sets of diagnoses.

2.1. Description logics

Since the underlying knowledge representation method of ontologies in the Semantic Web is based on description logics, we start by briefly introducing the main concepts, employing the usual definitions as in [15, 16]. A knowledge base is comprised of two components, namely a TBox (denoted by $\mathcal{T}$) and a ABox ($\mathcal{A}$). The TBox defines the terminology whereas the ABox contains assertions about named individuals in terms of the vocabulary defined in the TBox. The vocabulary consists of concepts, denoting sets of individuals, and roles, denoting binary relationships between individuals. These concepts and roles may be either

atomic or complex, the latter being obtained by employing description operators. The language of descriptions is defined recursively by starting from a schema $S = (\mathcal{CN}, \mathcal{RN}, \mathcal{IN})$ of disjoint sets of names for concepts, roles, and individuals. Typical operators for the construction of complex descriptions are $C \sqcup D$ (disjunction), $C \sqcap D$ (conjunction), $\neg C$ (negation), $\forall R.C$ (concept value restriction), and $\exists R.C$(concept exists restriction), where C and D are elements of $\mathcal{CN}$ and $R \in \mathcal{RN}$.

Knowledge bases are defined by a finite set of logical sentences. Sentences regarding the TBox are called terminological axioms whereas sentences regarding the ABox are called assertional axioms. Terminological axioms are expressed by $C \sqsubseteq D$ (Generalized Concept Inclusion) which corresponds to the logical implication. Let $a, b \in \mathcal{IN}$ be individual names. $C(a)$ and $R(a, b)$ are thus assertional axioms.

Concepts (rsp. roles) can be regarded as unary (rsp. binary) predicates. Roughly speaking description logics can be seen as fragments of first-order predicate logic (without considering transitive closure or special fixpoint semantics). These fragments are specifically designed to ensure decidability or favorable computational costs.

The semantics of description terms are usually given using an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a domain (non-empty universe) of values, and $(\cdot)^{\mathcal{I}}$ is a function that maps every concept description to a subset of $\Delta^{\mathcal{I}}$, and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The mapping also associates a value in $\Delta^{\mathcal{I}}$ with every individual name in $\mathcal{IN}$. An interpretation $\mathcal{I}$ is a model of a knowledge base iff it satisfies all terminological axioms and assertional axioms. A knowledge base is satisfiable iff a model exists. A concept description C is coherent (satisfiable) w.r.t. a TBox $\mathcal{T}$, if a model $\mathcal{I}$ of $\mathcal{T}$ exists such that $C^{\mathcal{I}} \neq \emptyset$. A TBox is incoherent iff an incoherent concept description exists.

2.2. Diagnosis of ontologies

Example 1. *Consider a simple ontology $\mathcal{O}$ with the terminology $\mathcal{T}$:*

$$ax_1 : A \sqsubseteq B \quad ax_2 : B \sqsubseteq C$$
$$ax_3 : C \sqsubseteq D \quad ax_4 : D \sqsubseteq R$$

and assertions $\mathcal{A} : \{A(w), \neg R(w), A(v)\}$.

Assume that the user explicitly states that the three assertional axioms should be considered as correct, i.e. these axioms are added to a background theory $\mathcal{B}$. The introduction of a background theory ensures that the diagnosis method focuses purely on the potentially faulty axioms.

Furthermore, assume that the user requires the currently inconsistent ontology $\mathcal{O}$ to be consistent. The only irreducible set of axioms (minimal conflict set) that preserves the inconsistency is $CS : \langle ax_1, ax_2, ax_3, ax_4 \rangle$. That is, one has to modify or remove the axioms of at least one of the following diagnoses

$$\mathcal{D}_1 : [ax_1] \quad \mathcal{D}_2 : [ax_2] \quad \mathcal{D}_3 : [ax_3] \quad \mathcal{D}_4 : [ax_4]$$

to restore the consistency of the ontology. However, it is unclear which of the ontologies $\mathcal{O}_i = \mathcal{O} \setminus \mathcal{D}_i$ obtained by application of diagnoses from the set $\mathbf{D} : \{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$ is the target one.

Definition 1. *A target ontology $\mathcal{O}_t$ is a set of logical sentences characterized by a set of background axioms $\mathcal{B}$, a set of sets of logical sentences P that must be entailed by $\mathcal{O}_t$ and the set of sets of logical sentences N that must not be entailed by $\mathcal{O}_t$.*

A target ontology $\mathcal{O}_t$ must fulfill the following necessary requirements:

- *$\mathcal{O}_t$ must be satisfiable (optionally coherent)*
- *$\mathcal{B} \subseteq \mathcal{O}_t$*
- *$\mathcal{O}_t \models p \quad \forall p \in P$*
- *$\mathcal{O}_t \not\models n \quad \forall n \in N$*

Given $\mathcal{B}$, P, and N, an ontology $\mathcal{O}$ is faulty iff $\mathcal{O}$ does not fulfill all the necessary requirements of the target ontology.

Note that the approach presented in this paper can be used with any knowledge representation language for which there exists a sound and complete procedure to decide whether $\mathcal{O} \models ax$ and the entailment operator $\models$ is extensive, monotone and idempotent. For instance, these requirements are fulfilled by all subsets of OWL 2 which are interpreted under OWL Direct Semantics.

Definition 1 allows a user to identify the target diagnosis $\mathcal{D}_t$ by providing sufficient information about the target ontology in the sets $\mathcal{B}, P$ and N. For instance, if in Example 1 the user provides the information that $\mathcal{O}_t \models \{B(w)\}$ and $\mathcal{O}_t \not\models \{C(w)\}$, the debugger will return only one diagnosis, namely $\mathcal{D}_2$. Application of this diagnosis results in a consistent ontology $\mathcal{O}_2 = \mathcal{O} \setminus \mathcal{D}_2$ that entails $\{B(w)\}$ because of ax_1 and the assertion $A(w)$. In addition, $\mathcal{O}_2$ does not entail $\{C(w)\}$ since $\mathcal{O}_2 \cup \{\neg C(w)\}$ is consistent and, moreover, $\{\neg R(w), ax_4, ax_3\} \models \{\neg C(w)\}$. All other ontologies $\mathcal{O}_i = (\mathcal{O} \setminus \mathcal{D}_i)$ obtained by the application of the diagnoses $\mathcal{D}_1, \mathcal{D}_3$ and $\mathcal{D}_4$ do not fulfill the given requirements, since $\mathcal{O}_1 \cup \{B(w)\}$ is inconsistent and therefore any consistent extension of $\mathcal{O}_1$ cannot entail $\{B(w)\}$. As both $\mathcal{O}_3$ and $\mathcal{O}_4$ entail $\{C(w)\}$, $\mathcal{O}_2$ corresponds to the target diagnosis $\mathcal{O}_t$.

Definition 2. *Let $\langle O, \mathcal{B}, P, N\rangle$ be a diagnosis problem instance, where O is an ontology, $\mathcal{B}$ a background theory, P a set of sets of logical sentences which must be entailed by the target ontology O_t, and N a set of sets of logical sentences which must* not *be entailed by O_t.*

A set of axioms $\mathcal{D} \subseteq O$ is a diagnosis iff the set of axioms $O \setminus \mathcal{D}$ can be extended by a logical description EX such that:

1. $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ *is consistent (and coherent if required)*
2. $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p \quad \forall p \in P$
3. $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n \quad \forall n \in N$

A diagnosis $\mathcal{D}_i$ defines a partition of the ontology O where each axiom $ax_j \in \mathcal{D}_i$ is a candidate for changes by the user and each axiom $ax_k \in O \setminus \mathcal{D}_i$ is correct. If $\mathcal{D}_t$ is the set of axioms of O to be changed (i.e. $\mathcal{D}_t$ is the target diagnosis) then the target ontology O_t is $(O \setminus \mathcal{D}_t) \cup \mathcal{B} \cup EX$ for some EX defined by the user.

In the following we assume the background theory $\mathcal{B}$ together with the sets of logical sentences in the sets P and N always allow formulation of the target ontology. Moreover, a diagnosis exists iff a target ontology exists.

Proposition 1. *A diagnosis $\mathcal{D}$ for a diagnosis problem instance $\langle O, \mathcal{B}, P, N\rangle$ exists iff*

$$\mathcal{B} \cup \bigcup_{p \in P} p$$

is consistent (coherent) and

$$\forall n \in N \; : \; \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$$

The set of all diagnoses is complete in the sense that at least one diagnosis exists where the ontology resulting from the trivial application of a diagnosis is a subset of the target ontology:

Proposition 2. *Let $\mathbf{D} \neq \emptyset$ be the set of all diagnoses for a diagnosis problem instance $\langle O, \mathcal{B}, P, N\rangle$ and O_t the target ontology. Then a diagnosis $\mathcal{D}_t \in \mathbf{D}$ exists s.t. $(O \setminus \mathcal{D}_t) \subseteq O_t$.*

The set of all diagnoses can be characterized by the set of minimal diagnoses.

Definition 3. *A diagnosis $\mathcal{D}$ for a diagnosis problem instance $\langle O, \mathcal{B}, P, N\rangle$ is a* minimal diagnosis *iff there is no $\mathcal{D}' \subset \mathcal{D}$ such that $\mathcal{D}'$ is a diagnosis.*

Proposition 3. *Let $\langle O, \mathcal{B}, P, N\rangle$ be a diagnosis problem instance. For every diagnosis $\mathcal{D}$ there is a minimal diagnosis $\mathcal{D}'$ s.t. $\mathcal{D}' \subseteq \mathcal{D}$.*

Definition 4. *A diagnosis $\mathcal{D}$ for a diagnosis problem instance $\langle O, \mathcal{B}, P, N\rangle$ is a* minimum cardinality diagnosis *iff there is no diagnosis $\mathcal{D}'$ such that $|\mathcal{D}'| < |\mathcal{D}|$.*

To summarize, a diagnosis describes which axioms are candidates for modification. Despite the fact that multiple diagnoses may exist, some are more preferable than others. E.g. minimal diagnoses require minimal changes, i.e. axioms are not considered for modification unless there is a reason. Minimal cardinality diagnoses require changing a minimal number of axioms. The actual type of error contained in an axiom is irrelevant as the concept of diagnosis defined here does not make any assumptions about errors themselves. There can, however, be instances where an ontology is faulty and the empty diagnosis is the only minimal diagnosis, e.g. if some axioms are missing and nothing must be changed.

The extension EX plays an important role in the ontology repair process, suggesting axioms that should be added to the ontology. For instance, in Example 1 the user requires that the target ontology *must not* entail $\{B(w)\}$ but has to entail $\{B(v)\}$, that is $N = \{\{B(w)\}\}$ and $P = \{\{B(v)\}\}$. Because, the example ontology O is inconsistent some sentences must be changed. The consistent ontology $O_1 = O \setminus \mathcal{D}_1$, neither entails $\{B(v)\}$ nor $\{B(w)\}$ (in particular $O_1 \models \{\neg B(w)\}$). Consequently, O_1 has to be extended with a set EX of logical sentences in order to entail $\{B(v)\}$. This set of logical sentences can be approximated with $EX = \{B(v)\}$. $O_1 \cup EX$ is satisfiable, entails $\{B(v)\}$ but does not entail $\{B(w)\}$. All other ontologies $O_i = O \setminus \mathcal{D}_i,\ i = 2, 3, 4$ are consistent but entail $\{B(w), B(v)\}$ and must be rejected because of the monotonic semantics of description logic. That is, there is no such extension EX that $(O_i \cup EX) \not\models \{B(w)\}$. Therefore, the diagnosis $\mathcal{D}_1$ is the minimum cardinality diagnosis which allows the formulation of the target ontology. Note that formulation of the complete extension is impossible, since our diagnosis approach deals with changes to existing axioms and does not learn new axioms.

The following corollary characterizes diagnoses without employing the true extension EX to formulate the target ontology. The idea is to use the sentences which must be entailed by the target ontology to approximate EX as shown above.

Corollary 1. *Given a diagnosis problem instance*

$\langle O, \mathcal{B}, P, N\rangle$, a set of axioms $\mathcal{D} \subseteq O$ is a diagnosis iff

$$(O \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p \qquad \textit{(Condition 1)}$$

is satisfiable (coherent) and

$$\forall n \in N \ : \ (O \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n \qquad \textit{(Condition 2)}$$

Proof sketch: ($\Rightarrow$) Let $\mathcal{D} \subseteq O$ be a diagnosis for $\langle O, \mathcal{B}, P, N\rangle$. Since there is an EX s.t. $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ is satisfiable (coherent) and $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p$ for all $p \in P$, it follows that $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \cup \bigcup_{p \in P} p$ is satisfiable (coherent) and therefore $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p$ is satisfiable (coherent). Consequently, the first condition of the corollary is fulfilled. Since $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \models p$ for all $p \in P$ and $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n$ for all $n \in N$ it follows that $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \cup \bigcup_{p \in P} p \not\models n$ for all $n \in N$. Consequently, $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$ for all $n \in N$ and the second condition of the corollary is fulfilled.

($\Leftarrow$) Let $\mathcal{D} \subseteq O$ and $\langle O, \mathcal{B}, P, N\rangle$ be a diagnosis problem instance. Without limiting generality let $EX = P$. By Condition 1 of the corollary $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup \bigcup_{p \in P} p$ is satisfiable (coherent). Therefore, for $EX = P$ the sentences $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX$ are satisfiable (coherent), i.e. the first condition for a diagnosis is fulfilled and these sentences entail p for all $p \in P$ which corresponds to the second condition a diagnosis must fulfill. Furthermore, by Condition 2 of the corollary $(O \setminus \mathcal{D}) \cup \mathcal{B} \cup EX \not\models n$ for all $n \in N$ holds and therefore the third condition for a diagnosis is fulfilled. Consequently, $\mathcal{D} \subseteq O$ is a diagnosis for $\langle O, \mathcal{B}, P, N\rangle$. □

Conflict sets, which are the parts of the ontology that preserve the inconsistency/incoherency, are usually employed to constrain the search space during computation of diagnoses.

Definition 5. *Given a diagnosis problem instance $\langle O, \mathcal{B}, P, N\rangle$, a set of axioms $CS \subseteq O$ is a conflict set iff $CS \cup \mathcal{B} \cup \bigcup_{p \in P} p$ is inconsistent (incoherent) or $n \in N$ exists s.t. $CS \cup \mathcal{B} \cup \bigcup_{p \in P} p \models n$.*

Definition 6. *A conflict set CS for an instance $\langle O, \mathcal{B}, P, N\rangle$ is minimal iff there is no $CS' \subset CS$ such that CS' is a conflict set.*

A set of minimal conflict sets can be used to compute the set of minimal diagnoses as shown in [17]. The idea is that each diagnosis must include at least one element of each minimal conflict set.

Proposition 4. *$\mathcal{D}$ is a minimal diagnosis for the diagnosis problem instance $\langle O, \mathcal{B}, P, N\rangle$ iff $\mathcal{D}$ is a minimal hitting set for the set of all minimal conflict sets of $\langle O, \mathcal{B}, P, N\rangle$.*

Ontology	Entailments
O_1	$\emptyset$
O_2	$\{B(w)\}$
O_3	$\{B(w), C(w)\}$
O_4	$\{B(w), C(w), D(w)\}$

Table 1: Entailments of ontologies $O_i = (O \setminus \mathcal{D}_i)$, $i = 1, \ldots, 4$ in Example 1 returned by realization.

Given a set of sets $\overline{S}$, a set H is a hitting set of $\overline{S}$ iff $H \cap S_i \neq \emptyset$ for all $S_i \in \overline{S}$ and $H \subseteq \bigcup_{S_i \in \overline{S}} S_i$. Most modern ontology diagnosis methods [7, 8, 9, 10] are implemented according to Proposition 4 and differ only in details, such as how and when (minimal) conflict sets are computed, the order in which hitting sets are generated, etc.

2.3. Differentiating between diagnoses

The diagnosis method usually generates a set of diagnoses for a given diagnosis problem instance. Thus, in Example 1 an ontology debugger returns a set of four minimal diagnoses $\{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$. As explained in the previous section, additional information, i.e. sets of sets of logical sentences P and N, can be used by the debugger to reduce the set of diagnoses. However, in the general case the user does not know which sets P and N to provide to the debugger such that the target diagnosis will be identified. Therefore, the debugger should be able to identify sets of logical sentences on its own and only ask the user or some other oracle, whether these sentences *must* or *must not* be entailed by the target ontology. To generate these sentences the debugger can apply each of the diagnoses in $\mathbf{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ and obtain a set of ontologies $O_i = O \setminus \mathcal{D}_i$, $i = 1, \ldots, n$ that fulfill the user requirements. For each ontology O_i a description logic reasoner can generate a set of entailments such as entailed subsumptions provided by the classification service and sets of class assertions provided by the realization service. These entailments can be used to discriminate between the diagnoses, as different ontologies entail different sets of sentences due to extensivity of the entailment relation. Note that in the examples provided in this section we consider only two types of entailments, namely subsumption and class assertion. In general, the approach presented in this paper is not limited to these types and can use all of the entailment types supported by a reasoner.

For instance, in Example 1 for each ontology $O_i = (O \setminus \mathcal{D}_i)$, $i = 1 \ldots 4$ the realization service of a reasoner returns the set of class assertions presented in Table 1. Without any additional information the debugger cannot decide which of these sentences must be entailed

by the target ontology. To obtain this information the diagnosis method must query an oracle that can specify whether the target ontology entails some set of sentences or not. E.g. the debugger could ask an oracle if $\{D(w)\}$ is entailed by the target ontology ($\mathcal{O}_t \models \{D(w)\}$). If the answer is *yes*, then $\{D(w)\}$ is added to P and $\mathcal{D}_4$ is considered as the target diagnosis. All other diagnoses are rejected because $(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \{D(w)\}$ for $i = 1, 2, 3$ is inconsistent. If the answer is *no*, then $\{D(w)\}$ is added to N and $\mathcal{D}_4$ is rejected as $(\mathcal{O} \setminus \mathcal{D}_4) \cup \mathcal{B} \models \{D(w)\}$ and we have to ask the oracle another question. In the following we consider a query Q as a set of logical sentences such that $\mathcal{O}_t \models Q$ holds iff $\mathcal{O}_t \models q_i$ for all $q_i \in Q$.

Property 1. *Given a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$, a set of diagnoses $\mathbf{D}$, a set of logical sentences Q representing the query ($\mathcal{O}_t \models Q$) and an oracle able to evaluate the query:*

If the oracle answers yes *then every diagnosis $\mathcal{D}_i \in \mathbf{D}$ is a diagnosis for $P \cup \{Q\}$ iff both conditions hold:*

$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \cup Q \text{ is consistent (coherent)}$$
$$\forall n \in N \ : \ (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \cup Q \not\models n$$

If the oracle answers no *then every diagnosis $\mathcal{D}_i \in \mathbf{D}$ is a diagnosis for $N \cup \{Q\}$ iff both conditions hold:*

$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \text{ is consistent (coherent)}$$
$$\forall n \in (N \cup \{Q\}) \ : \ (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \not\models n$$

In particular, a query partitions the set of diagnoses $\mathbf{D}$ into three disjoint subsets.

Definition 7. *For a query Q, each diagnosis $\mathcal{D}_i \in \mathbf{D}$ of a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ can be assigned to one of the three sets $\mathbf{D^P}$, $\mathbf{D^N}$ or $\mathbf{D}^\emptyset$ where*

- $\mathcal{D}_i \in \mathbf{D^P}$ *iff it holds that*
$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \models Q$$
- $\mathcal{D}_i \in \mathbf{D^N}$ *iff it holds that*
$$(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup \bigcup_{p \in P} p \cup Q$$
is inconsistent (incoherent).
- $\mathcal{D}_i \in \mathbf{D}^\emptyset$ *iff* $\mathcal{D}_i \in \mathbf{D} \setminus \left(\mathbf{D^P} \cup \mathbf{D^N}\right)$

Given a diagnosis problem instance we say that the diagnoses in $\mathbf{D^P}$ predict a positive answer (*yes*) as a result of the query Q, diagnoses in $\mathbf{D^N}$ predict a negative answer (*no*), and diagnoses in $\mathbf{D}^\emptyset$ do not make any predictions.

Property 2. *Given a diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$, a set of diagnoses $\mathbf{D}$, a query Q and an oracle:*

If the oracle answers yes *then the set of rejected diagnoses is $\mathbf{D^N}$ and the set of remaining diagnoses is $\mathbf{D^P} \cup \mathbf{D}^\emptyset$.*

If the oracle answers no *then the set of rejected diagnoses is $\mathbf{D^P}$ and the set of remaining diagnoses is $\mathbf{D^N} \cup \mathbf{D}^\emptyset$.*

Consequently, given a query Q either $\mathbf{D^P}$ or $\mathbf{D^N}$ is eliminated but $\mathbf{D}^\emptyset$ always remains after the query is answered. For generating queries we have to investigate for which subsets $\mathbf{D^P}, \mathbf{D^N} \subseteq \mathbf{D}$ a query exists that can differentiate between these sets. A straight forward approach is to investigate all possible subsets of $\mathbf{D}$. In our evaluation we show that this is feasible if we limit the number n of minimal diagnoses to be considered during query generation and selection. E.g. for $n = 9$, the algorithm has to verify 512 possible partitions in the worst case.

Given a set of diagnoses $\mathbf{D}$ for the ontology $\mathcal{O}$, a set P of sets of sentences that must be entailed by the target ontology $\mathcal{O}_t$ and a set of background axioms $\mathcal{B}$, the set of partitions $\mathbf{PR}$ for which a query exists can be computed as follows:

1. Generate the power set $\mathcal{P}(\mathbf{D})$, $\mathbf{PR} \leftarrow \emptyset$
2. Assign an element of $\mathcal{P}(\mathbf{D})$ to the set $\mathbf{D_i^P}$ and generate a set of common entailments E_i of all ontologies $(\mathcal{O} \setminus \mathcal{D}_j) \cup \mathcal{B} \cup \bigcup_{p \in P} p$, where $\mathcal{D}_j \in \mathbf{D_i^P}$
3. If $E_i = \emptyset$, then reject the current element $\mathbf{D_i^P}$, i.e. set $\mathcal{P}(\mathbf{D}) \leftarrow \mathcal{P}(\mathbf{D}) \setminus \{\mathbf{D_i^P}\}$ and goto Step 2. Otherwise set $Q_i \leftarrow E_i$.
4. Use Definition 7 and the query Q_i to classify the diagnoses $\mathcal{D}_k \in \mathbf{D} \setminus \mathbf{D_i^P}$ into the sets $\mathbf{D_i^P}$, $\mathbf{D_i^N}$ and $\mathbf{D_i^\emptyset}$. The generated partition is added to the set of partitions $\mathbf{PR} \leftarrow \mathbf{PR} \cup \{\langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle\}$ and set $\mathcal{P}(\mathbf{D}) \leftarrow \mathcal{P}(\mathbf{D}) \setminus \{\mathbf{D_i^P}\}$. If $\mathcal{P}(\mathbf{D}) \neq \emptyset$ then go to Step 2.

In Example 1 the set of diagnoses $\mathbf{D}$ of the ontology $\mathcal{O}$ contains 4 elements. Therefore, the power set $\mathcal{P}(\mathbf{D})$ includes 15 elements $\{\{\mathcal{D}_1\}, \{\mathcal{D}_2\}, \ldots, \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}\}$, assuming we omit the element corresponding to $\emptyset$ as it does not contain any diagnoses to be evaluated. Moreover, assume that P and N are empty. In each iteration an element of $\mathcal{P}(\mathbf{D})$ is assigned to the set $\mathbf{D_i^P}$.

For instance, the algorithm assigns $\mathbf{D_1^P} = \{\mathcal{D}_1, \mathcal{D}_2\}$. In this case the set of common entailments is empty as $(\mathcal{O} \setminus \mathcal{D}_1) \cup \mathcal{B}$ has no entailed sentences (see Table 1). Therefore, the set $\{\mathcal{D}_1, \mathcal{D}_2\}$ is rejected and removed from $\mathcal{P}(\mathbf{D})$. Assume that in the next iteration the algorithm selects $\mathbf{D_2^P} = \{\mathcal{D}_2, \mathcal{D}_3\}$. In this case the set of common entailments $E_2 = \{B(w)\}$ is not empty and so $Q_2 = \{B(w)\}$. The remaining diagnoses $\mathcal{D}_1$ and $\mathcal{D}_4$ are classified according to Definition 7. That is, the algorithm selects the first diagnosis $\mathcal{D}_1$ and verifies whether $(\mathcal{O} \setminus \mathcal{D}_1) \cup \mathcal{B} \models \{B(w)\}$. Given the negative answer of the reasoner, the algorithm checks if $(\mathcal{O} \setminus \mathcal{D}_1) \cup \mathcal{B} \cup \{B(w)\}$ is inconsistent. Since the condition is satisfied the diagnosis $\mathcal{D}_1$ is added to the set $\mathbf{D_2^N}$. The second diagnosis $\mathcal{D}_4$ is added to the set $\mathbf{D_2^P}$ as it satisfies the first requirement $(\mathcal{O} \setminus \mathcal{D}_4) \cup \mathcal{B} \models \{B(w)\}$. The resulting partition $\langle\{B(w)\}, \{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}, \{\mathcal{D}_1\}, \emptyset\rangle$ is added to the set **PR**.

However, a query need not include all of the entailed sentences. If a query Q partitions the set of diagnoses into $\mathbf{D^P}$, $\mathbf{D^N}$ and $\mathbf{D^\emptyset}$ and an (irreducible) subset $Q' \subset Q$ exists which preserves the partition then it is sufficient to query Q'. In our example, $Q_2 : \{B(w), C(w)\}$ can be reduced to its subset $Q_2' : \{C(w)\}$. If there are multiple irreducible subsets that preserve the partition then we select one of them.

All of the queries and their corresponding partitions generated in Example 1 are presented in Table 2. Given these queries the debugger has to decide which one should be asked first in order to minimize the number of queries to be answered. A popular query selection heuristic (called "split-in-half") prefers queries which allow half of the diagnoses to be removed from the set **D** regardless of the answer of an oracle.

Using the data presented in Table 2, the "split-in-half" heuristic determines that asking the oracle if $(\mathcal{O}_t \models \{C(w)\})$ is the best query (i.e. the reduced query Q_2), as two diagnoses from the set **D** are removed regardless of the answer. Assuming that $\mathcal{D}_1$ is the target diagnosis, then an oracle will answer *no* to our question (i.e. $\mathcal{O}_t \not\models \{C(w)\}$). Based on this feedback, the diagnoses $\mathcal{D}_3$ and $\mathcal{D}_4$ are removed according to Property 2. Given the updated set of diagnoses **D** and $P = \{\{C(w)\}\}$ the partitioning algorithm returns the only partition $\langle\{B(w)\}, \{\mathcal{D}_2\}, \{\mathcal{D}_1\}, \emptyset\rangle$. The heuristic then selects the query $\{B(w)\}$, which is also answered with *no* by the oracle. Consequently, $\mathcal{D}_1$ is identified as the only remaining minimal diagnosis.

In general, if n is the number of diagnoses and we can split the set of diagnoses in half with each query, then the minimum number of queries is $log_2 n$. Note that this minimum number of queries can only be achieved when all minimal diagnoses are considered at once, which is intractable even for relatively small values of n.

However, in case probabilities of diagnoses are known we can reduce the number of queries by utilizing two effects:

1. We can exploit diagnoses probabilities to assess the likelihood of each answer and the expected value of the information contained in the set of diagnoses after an answer is given.
2. Even if multiple diagnoses remain, further query generation may not be required if one diagnosis is highly probable and all other remaining diagnoses are highly improbable.

Example 2. *Consider an ontology $\mathcal{O}$ with the terminology $\mathcal{T}$:*

$$ax_1 : A_1 \sqsubseteq A_2 \sqcap M_1 \sqcap M_2 \qquad ax_4 : M_2 \sqsubseteq \forall s.A \sqcap D$$
$$ax_2 : A_2 \sqsubseteq \neg\exists s.M_3 \sqcap \exists s.M_2 \qquad ax_5 : M_3 \equiv B \sqcup C$$
$$ax_3 : M_1 \sqsubseteq \neg A \sqcap B$$

and the background theory containing the assertions $\mathcal{A} : \{A_1(w), A_1(u), s(u, w)\}$.

The ontology is inconsistent and the set of minimal conflict sets $CS = \{\langle ax_1, ax_3, ax_4\rangle, \langle ax_1, ax_2, ax_3, ax_5\rangle\}$. To restore consistency, the user should modify all axioms of at least one minimal diagnosis:

$$\mathcal{D}_1 : [ax_1] \qquad \mathcal{D}_3 : [ax_4, ax_5]$$
$$\mathcal{D}_2 : [ax_3] \qquad \mathcal{D}_4 : [ax_4, ax_2]$$

Following the same approach as in the first example, we compute a set of possible queries and corresponding partitions using the algorithm presented above. A set of possible irreducible queries for Example 2 and their partitions are presented in Table 3. These queries partition the set of diagnoses **D** in a way that makes the application of myopic strategies, such as "split-in-half", inefficient. A greedy algorithm based on such a heuristic would first select the first query Q_1, since there is no query that cuts the set of diagnoses in half. If $\mathcal{D}_4$ is the target diagnosis then Q_1 will be answered with *yes* by an oracle (see Figure 1). In the next iteration the algorithm would also choose a suboptimal query, the first untried query Q_2, since there is no partition that divides the diagnoses $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_4$ into two groups of equal size. Once again, the oracle answers *yes*, and the algorithm identifies query Q_4 to differentiate between $\mathcal{D}_1$ and $\mathcal{D}_4$.

However, in real-world settings the assumption that all axioms fail with the same probability is rarely the case. For example, Roussey et al. [6] present a list of "anti-patterns" where an anti-pattern is a set of axioms, such as $\{C1 \sqsubseteq \forall R.C2, C1 \sqsubseteq \forall R.C3, C2 \equiv \neg C3\}$ that

Query	$\mathbf{D^P}$	$\mathbf{D^N}$	$\mathbf{D^\emptyset}$
$Q_1 : \{B(w)\}$	$\{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$	$\emptyset$
$Q_2 : \{B(w), C(w)\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_2\}$	$\emptyset$
$Q_3 : \{B(w), C(w), Q(w)\}$	$\{\mathcal{D}_4\}$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$	$\emptyset$

Table 2: Possible queries in Example 1

Query	$\mathbf{D^P}$	$\mathbf{D^N}$	$\mathbf{D^\emptyset}$
$Q_1 : \{B \sqsubseteq M_3\}$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4\}$	$\{\mathcal{D}_3\}$	$\emptyset$
$Q_2 : \{B(w)\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_2\}$	$\{\mathcal{D}_1\}$
$Q_3 : \{M_1 \sqsubseteq B\}$	$\{\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_2\}$	$\emptyset$
$Q_4 : \{M_1(w), M_2(u)\}$	$\{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$	$\emptyset$
$Q_5 : \{A(w)\}$	$\{\mathcal{D}_2\}$	$\{\mathcal{D}_3, \mathcal{D}_4\}$	$\{\mathcal{D}_1\}$
$Q_6 : \{M_2 \sqsubseteq D\}$	$\{\mathcal{D}_1, \mathcal{D}_2\}$	$\emptyset$	$\{\mathcal{D}_3, \mathcal{D}_4\}$
$Q_7 : \{M_3(u)\}$	$\{\mathcal{D}_4\}$	$\emptyset$	$\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$

Table 3: Possible queries in Example 2

corresponds to a minimal conflict set. The study performed by [6] shows that such conflict sets often occur in practice due to frequent misuse of certain language constructs like quantification or disjointness. Such studies are ideal sources for estimating prior fault probabilities. However, this is beyond the scope of this paper.

Our approach for computing the prior fault probabilities of axioms is inspired by Rector et al. [5] and considers the syntax of a knowledge representation language, such as restrictions, conjunction, negation, etc. For instance, if a user frequently changes the universal to the existential quantifier and vice versa in order to restore coherency, then we can assume that axioms including such restrictions are more likely to fail than the other ones. In [5] the authors report that in most cases inconsistent ontologies are created because users (a) mix up $\forall r.S$ and $\exists r.S$, (b) mix up $\neg\exists r.S$ and $\exists r.\neg S$, (c) mix up $\sqcup$ and $\sqcap$, (d) wrongly assume that classes are disjoint by default or overuse disjointness, or (e) wrongly apply negation. Observing that misuses of quantifiers are more likely than other failure patterns one might find that the axioms ax_2 and ax_4 are more likely to be faulty

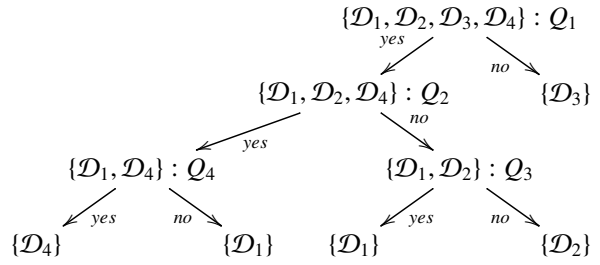

Figure 1: The search tree of the greedy algorithm

than ax_3 (because of the use of quantifiers), whereas ax_3 is more likely to be faulty than ax_5 and ax_1 (because of the use of negation).

Detailed justifications of diagnoses probabilities are given in the next section. However, let us assume some probability distribution of the faults according to the observations presented above such that: (a) the diagnosis $\mathcal{D}_2$ is the most probable one, i.e. single fault diagnosis of an axiom containing a negation; (b) although $\mathcal{D}_4$ is a double fault diagnosis, it follows $\mathcal{D}_2$ closely as its axioms contain quantifiers; (c) $\mathcal{D}_1$ and $\mathcal{D}_3$ are significantly less probable than $\mathcal{D}_4$ because conjunction/disjunction in ax_1 and ax_5 have a significantly lower fault probability than negation in ax_3. Taking this information into account asking query Q_1 is essentially useless because it is highly probable that the target diagnosis is either $\mathcal{D}_2$ or $\mathcal{D}_4$ and, therefore, it is highly probable that the oracle will respond with *yes*. Instead, asking Q_3 is more informative because regardless of the answer we can exclude one of the highly probable diagnoses, i.e. either $\mathcal{D}_2$ or $\mathcal{D}_4$. If the oracle responds to Q_3 with *no* then $\mathcal{D}_2$ is the only remaining diagnosis. However, if the oracle responds with *yes*, diagnoses $\mathcal{D}_4$, $\mathcal{D}_3$, and $\mathcal{D}_1$ remain, where $\mathcal{D}_4$ is significantly more probable compared to diagnoses $\mathcal{D}_3$ and $\mathcal{D}_1$. If the difference between the probabilities of the diagnoses is high enough such that $\mathcal{D}_4$ can be accepted as the target diagnosis, no additional questions are required. Obviously this strategy can lead to a substantial reduction in the number of queries compared to myopic approaches as we demonstrate in our evaluation.

Note that in real-world application scenarios failure patterns and their probabilities can be discovered by analyzing the debugging actions of a user in an ontology

editor, like Protégé. Learning of fault probabilities can be used to "personalize" the query selection algorithm to prefer user-specific faults. However, as our evaluation shows, even a rough estimate of the probabilities is capable of outperforming the "split-in-half" heuristic.

3. Entropy-based query selection

To select the best query we exploit a-priori failure probabilities of each axiom derived from the syntax of description logics or some other knowledge representation language, such as OWL. That is, the user is able to specify own beliefs in terms of the probability of syntax element such as $\forall$, $\exists$, $\sqcap$, etc. being erroneous; alternatively, the debugger can compute these probabilities by analyzing the frequency of various syntax elements in the target diagnoses of different debugging sessions. If no failure information is available then the debugger can initialize all of the probabilities with some small value. Compared to statistically well-founded probabilities, the latter approach provides a suboptimal but useful diagnosis discrimination process, as discussed in the evaluation.

Given the failure probabilities of all syntax elements $se \in \mathbf{S}$ of a knowledge representation language used in $\mathcal{O}$, we can compute the failure probability of an axiom $ax_i \in \mathcal{O}$

$$p(ax_i) = p(F_{se_1} \cup F_{se_2} \cup \cdots \cup F_{se_n})$$

where $F_{se_1} \ldots F_{se_n}$ represent the events that the occurrence of a syntax element se_j in ax_i is faulty. E.g. for ax_2 of Example 2 $p(ax_2) = p(F_{\sqsubseteq} \cup F_{\neg} \cup F_{\exists} \cup F_{\sqcap} \cup F_{\exists})$. Assuming that each occurrence of a syntax element fails independently, i.e. an erroneous usage of a syntax element se_k makes it neither more nor less probable that an occurrence of syntax element se_j is faulty, the failure probability of an axiom is computed as:

$$p(ax_i) = 1 - \prod_{se \in \mathbf{S}} (1 - F_{se})^{c(se)} \tag{1}$$

where $c(se_j)$ returns number of occurrences of the syntax element se_j in an axiom ax_i. If among other failure probabilities the user states that $p(F_{\sqsubseteq}) = 0.001$, $p(F_{\neg}) = 0.01$, $p(F_{\exists}) = 0.05$ and $p(F_{\sqcap}) = 0.001$ then $p(ax_2) = p(F_{\sqsubseteq} \cup F_{\neg} \cup F_{\exists} \cup F_{\sqcap} \cup F_{\exists}) = 0.108$.

Given the failure probabilities $p(ax_i)$ of axioms, the diagnosis algorithm first calculates the a-priori probability $p(\mathcal{D}_j)$ that $\mathcal{D}_j$ is the target diagnosis. Since all axioms fail independently, this probability can be computed as [18]:

$$p(\mathcal{D}_j) = \prod_{ax_n \in \mathcal{D}_j} p(ax_n) \prod_{ax_m \in \mathcal{O} \setminus \mathcal{D}_j} 1 - p(ax_m) \tag{2}$$

The prior probabilities for diagnoses are then used to initialize an iterative algorithm that includes two main steps: (a) the selection of the best query and (b) updating the diagnoses probabilities given query feedback.

According to information theory the best query is the one that, given the answer of an oracle, minimizes the expected entropy of the set of diagnoses [18]. Let $p(Q_i = yes)$ be the probability that query Q_i is answered with *yes* and $p(Q_i = no)$ be the probability for the answer *no*. Furthermore, let $p(\mathcal{D}_j|Q_i = yes)$ be the probability of diagnosis $\mathcal{D}_j$ after the oracle answers *yes* and $p(\mathcal{D}_j|Q_i = no)$ be the probability after the oracle answers *no*. The expected entropy after querying Q_i is:

$$H_e(Q_i) = \sum_{v \in \{yes,no\}} p(Q_i = v) \times \\ - \sum_{\mathcal{D}_j \in \mathbf{D}} p(\mathcal{D}_j|Q_i = v) \log_2 p(\mathcal{D}_j|Q_i = v)$$

Based on a one-step-look-ahead information theoretic measure, the query which minimizes the expected entropy is considered best. This formula can be simplified to the following score function [18] which we use to evaluate all available queries and select the one with the minimum score to maximize information gain:

$$sc(Q_i) = \sum_{v \in \{yes,no\}} [p(Q_i = v) \log_2 p(Q_i = v)] \\ + p(\mathbf{D}_\mathbf{i}^\emptyset) + 1 \tag{3}$$

where $v \in \{yes, no\}$ is a feedback of an oracle and $\mathbf{D}_\mathbf{i}^\emptyset$ is the set of diagnoses which do not make any predictions for the query Q_i. The probability of the set of diagnoses $p(\mathbf{D}_\mathbf{i}^\emptyset)$ as well as of any other set of diagnoses $\mathbf{D_i}$ like $\mathbf{D}_\mathbf{i}^\mathbf{P}$ and $\mathbf{D}_\mathbf{i}^\mathbf{N}$ is computed as:

$$p(\mathbf{D_i}) = \sum_{\mathcal{D}_j \in \mathbf{D_i}} p(\mathcal{D}_j)$$

because by Definition 2, each diagnosis uniquely partitions all of the axioms of an ontology $\mathcal{O}$ into two sets, correct and faulty, and thus all diagnoses are mutually exclusive events.

Since, for a query Q_i, the set of diagnoses $\mathbf{D}$ can be partitioned into the sets $\mathbf{D}_\mathbf{i}^\mathbf{P}$, $\mathbf{D}_\mathbf{i}^\mathbf{N}$ and $\mathbf{D}_\mathbf{i}^\emptyset$, the probability that an oracle will answer a query Q_i with either *yes* or *no* can be computed as:

$$p(Q_i = yes) = p(\mathbf{D}_\mathbf{i}^\mathbf{P}) + p(\mathbf{D}_\mathbf{i}^\emptyset)/2 \\ p(Q_i = no) = p(\mathbf{D}_\mathbf{i}^\mathbf{N}) + p(\mathbf{D}_\mathbf{i}^\emptyset)/2 \tag{4}$$

Clearly this assumes that for each diagnosis of $\mathbf{D}_\mathbf{i}^\emptyset$ *both outcomes are equally likely* and thus the probability that the set of diagnoses $\mathbf{D}_\mathbf{i}^\emptyset$ predicts either $Q_i = yes$ or $Q_i = no$ is $p(\mathbf{D}_\mathbf{i}^\emptyset)/2$.

Following feedback v for a query Q_s, i.e. $Q_s = v$, the probabilities of the diagnoses must be updated to take the new information into account. The update is made using Bayes' rule for each $\mathcal{D}_j \in \mathbf{D}$:

$$p(\mathcal{D}_j|Q_s = v) = \frac{p(Q_s = v|\mathcal{D}_j)p(\mathcal{D}_j)}{p(Q_s = v)} \tag{5}$$

where the denominator $p(Q_s = v)$ is known from the query selection step (Equation 4) and $p(\mathcal{D}_j)$ is either a prior probability (Equation 2) or is a probability calculated using Equation 5 after a previous iteration of the debugging algorithm. We assign $p(Q_s = v|\mathcal{D}_j)$ as follows:

$$p(Q_s = v|\mathcal{D}_j) = \begin{cases} 1, & \text{if } \mathcal{D}_j \text{ predicted } Q_s = v; \\ 0, & \text{if } \mathcal{D}_j \text{ is rejected by } Q_s = v; \\ \frac{1}{2}, & \text{if } \mathcal{D}_j \in \mathbf{D_s^\emptyset} \end{cases}$$

Example 1 (continued) Suppose that the debugger is not provided with any information about possible failures and therefore assumes that all syntax elements fail with the same probability 0.01 and therefore $p(ax_i) = 0.01$ for all $ax_i \in \mathcal{O}$. Using Equation 2 we can calculate probabilities for each diagnosis. For instance, $\mathcal{D}_1$ suggests that only one axiom ax_1 should be modified by the user. Hence, we can calculate the probability of diagnosis D_1 as $p(\mathcal{D}_1) = p(ax_1)(1 - p(ax_2))(1 - p(ax_3))(1 - p(ax_4)) = 0.0097$. All other minimal diagnoses have the same probability, since every other minimal diagnosis suggests the modification of one axiom. To simplify the discussion we only consider minimal diagnoses for query selection. Therefore, the prior probabilities of the diagnoses can be normalized to $p(\mathcal{D}_j) = p(\mathcal{D}_j)/\sum_{\mathcal{D}_j \in \mathbf{D}} p(\mathcal{D}_j)$ and are equal to 0.25.

Given the prior probabilities of the diagnoses and a set of queries (see Table 2) we evaluate the score function (Equation 3) for each query. E.g. for the first query $Q_1 : \{B(w)\}$ the probability $p(\mathbf{D}^\emptyset) = 0$ and the probabilities of both the positive and negative outcomes are: $p(Q_1 = 1) = p(\mathcal{D}_2) + p(\mathcal{D}_3) + p(\mathcal{D}_4) = 0.75$ and $p(Q_1 = 0) = p(\mathcal{D}_1) = 0.25$. Therefore the query score is $sc(Q_1) = 0.1887$.

The scores computed during the initial stage (see Table 4) suggest that Q_2 is the best query. Taking into account that $\mathcal{D}_1$ is the target diagnosis the oracle answers *no* to the query. The additional information obtained from the answer is then used to update the probabilities of diagnoses using the Equation 5. Since $\mathcal{D}_1$ and $\mathcal{D}_2$ predicted this answer, their probabilities are updated, $p(\mathcal{D}_1) = p(\mathcal{D}_2) = 1/p(Q_2 = 1) = 0.5$. The probabilities of diagnoses $\mathcal{D}_3$ and $\mathcal{D}_4$ which are rejected by the oracle's answer are also updated, $p(\mathcal{D}_3) = p(\mathcal{D}_4) = 0$.

Query	Initial score	$Q_2 = yes$
$Q_1 : \{B(w)\}$	0.1887	**0**
$Q_2 : \{C(w)\}$	**0**	1
$Q_3 : \{Q(w)\}$	0.1887	1

Table 4: Expected scores for minimized queries ($p(ax_i) = 0.01$)

Query	Initial score
$Q_1 : \{B(w)\}$	**0.250**
$Q_2 : \{C(w)\}$	0.408
$Q_3 : \{Q(w)\}$	0.629

Table 5: Expected scores for minimized queries ($p(ax_1) = 0.025$, $p(ax_2) = p(ax_3) = p(ax_4) = 0.01$)

In the next iteration the algorithm recomputes the scores using the updated probabilities. The results show that Q_1 is the best query. The other two queries Q_2 and Q_3 are irrelevant since no information will be gained if they are asked. Given the oracle's negative feedback to Q_1, we update the probabilities $p(\mathcal{D}_1) = 1$ and $p(\mathcal{D}_2) = 0$. In this case the target diagnosis $\mathcal{D}_1$ was identified using the same number of steps as the "split-in-half" heuristic.

However, if the user specifies that the first axiom is more likely to fail, e.g. $p(ax_1) = 0.025$, then $Q_1 : \{B(w)\}$ will be selected first (see Table 5). The recalculation of the probabilities given the negative outcome $Q_1 = 0$ sets $p(\mathcal{D}_1) = 1$ and $p(\mathcal{D}_2) = p(\mathcal{D}_3) = p(\mathcal{D}_4) = 0$. Therefore the debugger identifies the target diagnosis in only one step.

Example 2 (continued) Suppose that in ax_4 the user specified $\forall s.A$ instead of $\exists s.A$ and $\neg\exists s.M_3$ instead of $\exists s.\neg M_3$ in ax_2. Therefore $\mathcal{D}_4$ is the target diagnosis. Moreover, assume that the debugger is provided with observations of three types of faults: (1) conjunction/disjunction occurs with probability $p_1 = 0.001$, (2) negation $p_2 = 0.01$, and (3) restrictions $p_3 = 0.05$. Using Equation 1 we can calculate the probability of the axioms containing an error: $p(ax_1) = 0.0019$, $p(ax_2) = 0.1074$, $p(ax_3) = 0.012$, $p(ax_4) = 0.051$, and $p(ax_5) = 0.001$. These probabilities are exploited to calculate the prior probabilities of the diagnoses (see Table 6) and to initialize the query selection process. To simplify matters we focus on the set of minimal diagnoses.

In the first iteration the algorithm determines that Q_3 is the best query and asks the oracle whether $\mathcal{O}_t \models \{M_1 \sqsubseteq B\}$ is true or not (see Table 7). The obtained information is then used to recalculate the probabilities of the diagnoses and to compute the next best subsequent

query, i.e. Q_4, and so on. The query process stops after the third query, since $\mathcal{D}_4$ is the only diagnosis that has the probability $p(\mathcal{D}_4) > 0$.

Given the feedback of the oracle $Q_4 = yes$ for the second query, the updated probabilities of the diagnoses show that the target diagnosis has a probability of $p(\mathcal{D}_4) = 0.9918$ whereas $p(\mathcal{D}_3)$ is only 0.0082. In order to reduce the number of queries a user can specify a threshold, e.g. $\sigma = 0.95$. If the absolute difference in probabilities of two most probable diagnoses is greater than this threshold, the query process stops and returns the most probable diagnosis. Therefore, in this example the debugger based on the entropy query selection requires less queries than the "split-in-half" heuristic. Note that already after the first answer $Q_3 = yes$ the most probable diagnosis $\mathcal{D}_4$ is three times more likely than the second most probable diagnosis $\mathcal{D}_1$. Given such a great difference we could suggest to stop the query process after the first answer if the user would set $\sigma = 0.65$.

4. Implementation details

The iterative ontology debugger (Algorithm 1) takes a faulty ontology O as input. Optionally, a user can provide a set of axioms $\mathcal{B}$ that are known to be correct as well as a set P of axioms that must be entailed by the target ontology and a set N of axioms that must not. If these sets are not given, the corresponding input arguments are initialized with $\emptyset$. Moreover, the algorithm takes a set FP of fault probabilities for axioms $ax_i \in O$, which can be computed as described in Section 3 by exploiting knowledge about typical user errors. Alternatively, if no estimates of such probabilities are available, all probability values can be initialized using a small constant. We show the results of such a strategy in our evaluation section. The two other arguments σ and n are used to improve the performance of the algorithm. σ specifies the diagnosis acceptance threshold, i.e. the minimum difference in probabilities between the most likely and second-most likely diagnoses. The parameter n defines the maximum number of most probable diagnoses that should be considered by the algorithm during each iteration. A further performance gain in Algorithm 1 can be achieved if we approximate the set of the n most probable diagnoses with the set of the n most probable *minimal* diagnoses, i.e. we neglect non-minimal diagnoses. We call this set of at most n most probable minimal diagnoses the *leading diagnoses*. Note, under the reasonable assumption that the fault probability of each axiom $p(ax_i)$ is less than 0.5, for every non-minimal diagnosis ND a minimal diagnosis $\mathcal{D} \subset ND$ exists which from Equation 2 is more probable than ND. Consequently the query selection algorithm presented here operates on the set of minimal diagnoses instead of all diagnoses (i.e. non-minimal diagnoses are excluded). However, the algorithm can be adapted with moderate effort to also consider non-minimal diagnoses.

We use the approach proposed by Friedrich et al. [9] to compute diagnoses and employ the combination of two algorithms, QuickXplain [19] and HS-Tree [17]. In a standard implementation the latter is a breadth-first search algorithm that takes an ontology O, sets P and N, and the maximum number of most probable minimal diagnoses n as an input. The algorithm generates minimal hitting sets using minimal conflict sets, which are computed on-demand. This is motivated by the fact that in some circumstances a subset of all minimal conflict sets is sufficient for generating a subset of all required minimal diagnoses. For instance, in Example 2 the user wants to compute only $n = 2$ leading minimal diagnoses and a minimal conflict search algorithm returns CS_1. In this case HS-Tree identifies two required minimal diagnoses $\mathcal{D}_1$ and $\mathcal{D}_2$ and avoiding the computation of the minimal conflict set CS_2. Of course, in the worst case, when all minimal diagnoses have to be computed the algorithm should compute all minimal conflict sets. In addition, the HS-Tree generation reuses minimal conflict sets in order to avoid unnecessary computations. Thus, in the real-world scenarios we evaluated (see Table 8), less than 10 minimal conflict sets were contained in the faulty ontologies having at most 13 elements while the maximal cardinality of minimal diagnoses was observed to be at most 9. Therefore, space limitations were not a problem for the breadth-first generation. However, for scenarios involving diagnoses of greater cardinalities iterative-deepening strategies could be applied.

In our implementation of HS-Tree we use the uniform-cost search strategy. Given additional information in terms of axiom fault probabilities FP, the algorithm expands a leaf node in a search-tree if it is an element of the path corresponding to the maximum probability hitting set of minimal conflict sets computed so far. The probability of each minimal hitting set can be computed using Equation 2. Consequently, the algorithm computes a set of diagnoses ordered by their probability starting from the most probable one. HS-Tree terminates if either the n most probable minimal diagnoses are identified or no further minimal diagnoses can be found. Thus the algorithm computes at most n minimal diagnoses regardless of the number of all minimal diagnoses.

Answers	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
Prior	0.0970	0.5874	0.0026	0.3130
$Q_3 = yes$	0.2352	0	0.0063	0.7585
$Q_3 = yes$, $Q_4 = yes$	0	0	0.0082	0.9918
$Q_3 = yes$, $Q_4 = yes$, $Q_1 = yes$	0	0	0	1

Table 6: Probabilities of diagnoses after answers

Queries	Initial	$Q_3 = yes$	$Q_3 = yes$, $Q_4 = yes$
$Q_1 : \{B \sqsubseteq M_3\}$	0.974	0.945	**0.931**
$Q_2 : \{B(w)\}$	0.151	0.713	1
$Q_3 : \{M_1 \sqsubseteq B\}$	**0.022**	1	1
$Q_4 : \{M_1(w), M_2(u)\}$	0.540	**0.213**	1
$Q_5 : \{A(w)\}$	0.151	0.713	1
$Q_6 : \{M_2 \sqsubseteq D\}$	0.686	0.805	1
$Q_7 : \{M_3(u)\}$	0.759	0.710	0.970

Table 7: Expected scores for queries

HS-Tree uses QuickXplain to compute required minimal conflicts. This algorithm, given a set of axioms AX and a set of correct axioms $\mathcal{B}$ returns a minimal conflict set $CS \subseteq AX$, or $\emptyset$ if axioms $AX \cup \mathcal{B}$ are consistent. In the worst case, to compute a minimal conflict QuickXplain performs $2k(\log(s/k) + 1)$ consistency checks, where k is the size of the generated minimal conflict set and s is the number of axioms in the ontology. In the best case only $\log(s/k) + 2k$ are performed [19]. Importantly, the size of the ontology is contained in the log function. Therefore, the time needed for consistency checks in our test ontologies remained below 0.2 seconds, even for real world knowledge bases with thousands of axioms. The maximum time to compute a minimal conflict was observed in the Sweet-JPL ontology and took approx. 5 seconds (see Table 9).

In order to take past answers into account the HS-Tree updates the prior probabilities of the diagnoses by evaluating Equation 5. All required data is stored in the query history QH as well as in the sets P and N. When complete, HS-Tree returns a set of tuples of the form $\langle \mathcal{D}_i, p(\mathcal{D}_i) \rangle$ where $\mathcal{D}_i$ is contained in the set of the n most probable minimal diagnoses (leading diagnoses) and $p(\mathcal{D}_i)$ is its probability calculated using Equation 2 and Equation 5.

In the query-selection phase Algorithm 1 calls selectQuery function (Algorithm 2) to generate a tuple $T = \left\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset} \right\rangle$, where Q is the minimum score query (Equation 3) and $\mathbf{D^P}, \mathbf{D^N}$ and $\mathbf{D^\emptyset}$ the sets of diagnoses constituting the partition. The generation algorithm carries out a depth-first search, removing the top element of the set D and calling itself recursively to generate all possible subsets of the leading diagnoses. The set of leading diagnoses $\mathbf{D}$ is extracted from the set of tuples DP by the getDiagnoses function. In each leaf node of the search tree the generate function calls createQuery creates a query given a set of diagnoses $\mathbf{D^P}$ by computing common entailments and partitioning the set of diagnoses $\mathbf{D} \setminus \mathbf{D^P}$, as described in Section 2.3. If a query for the set $\mathbf{D^P}$ does not exist (i.e. there are no common entailments) or $\mathbf{D^P} = \emptyset$ then createQuery returns an empty tuple $T = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$. In all inner nodes

Algorithm 1: ontoDebugging($\mathcal{O}, \mathcal{B}, P, N, FP, n, \sigma$)

Input: ontology $\mathcal{O}$, set of background axioms $\mathcal{B}$, set of sets of logical sentences to be entailed P, set of sets of logical sentences not to be entailed N, set of fault probabilities for axioms FP, maximum number of most probable minimal diagnoses n, acceptance threshold σ

Output: a diagnosis $\mathcal{D}$

```
DP ← ∅; QH ← ∅; T ← ⟨∅, ∅, ∅, ∅⟩;
while belowThreshold(DP, σ) ∧ getScore(T) ≠ 1 do
    DP ← HS-Tree(O, B, P, N, FP, QH, n);
    T ← selectQuery(DP, O, B, P);
    Q ← getQuery(T);
    if Q = ∅ then exit loop;
    if getAnswer(O_t ⊨ Q) then P ← P ∪ {Q};
    else N ← N ∪ {Q};
    QH ← QH ∪ {T};
return mostProbableDiagnosis(DP);
```

Algorithm 2: selectQuery$(DP, \mathcal{O}, \mathcal{B}, P)$

Input: set DP of tuples $\langle \mathcal{D}_i, p(\mathcal{D}_i)\rangle$, ontology $\mathcal{O}$, set of background axioms $\mathcal{B}$, set of sets of logical sentences that must be entailed by the target ontology P

Output: a tuple $\left\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset}\right\rangle$

```
1  D ← getDiagnoses(DP);
2  T ← generate(∅, D, O, B, P, DP);
3  return minimizeQuery(T);
4  function generate (D^P, D, O, B, P, DP)
                 returns a tuple ⟨Q, D^P, D^N, D^∅⟩
5      if D = ∅ then
6          D ← getDiagnoses(DP);
7          return createQuery (D^P, O, B, P, D);
8      D ← pop (D);
9      left ← generate (D^P, D, O, B, P, DP);
10     right ← generate (D^P ∪ {D}, D, O, B, P, DP);
11     if getScore (left, DP) < getScore (right, DP)
       then return left;
12     else return right;
```

of the tree the algorithm selects a tuple that corresponds to a query with the minimum score as found using the getScore function. This function may implement the entropy-based measure (Equation 3), "split-in-half" or any other preference criteria. Given an empty tuple $T = \langle \emptyset, \emptyset, \emptyset, \emptyset\rangle$ the function returns the highest possible score of a used measure. In general, createQuery is called 2^n times, where we set $n = 9$ in our evaluation. Furthermore, for each leading diagnosis not in $\mathbf{D^P}$, createQuery has to check if the associated query is entailed. If a query is not entailed, a consistency check has to be performed. Entailments are determined by classification/realization and a subset check of the generated sentences. Common entailments are computed by exploiting the intersection of entailments for each diagnosis contained in $\mathbf{D^P}$. Note that the entailments for each leading diagnosis are computed just once and reused in for subsequent calls of createQuery.

In the function minimizeQuery, the query Q of the resulting tuple $\left\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset}\right\rangle$ is iteratively reduced by applying QuickXplain such that sets $\mathbf{D^P}$, $\mathbf{D^N}$ and $\mathbf{D^\emptyset}$ are preserved. This is implemented by replacing the consistency checks performed by QuickXplain with checks that ensure that the reduction of the query preserves the partition. In order to check if a partition is preserved, a consistency/entailment check is performed for each element in $\mathbf{D^N}$ and $\mathbf{D^\emptyset}$. Elements of $\mathbf{D^P}$ need not be checked because these elements entail the query and therefore any reduction. In the worst case $n(2k\log(s/k)+2k)$ consistency checks have to be performed in minimizeQuery where k is the length of the minimized query. Entailments of leading diagnoses are reused.

Algorithm 1 invokes the function getQuery to obtain the query from the tuple stored in T and calls getAnswer to query the oracle. Depending on the answer, Algorithm 1 extends either the set P or the set N and thus excludes diagnoses not compliant with the query answer from the results of HS-Tree in further iterations. Note, the algorithm can be easily adapted to allow the oracle to reject a query if the answer is unknown. In this case the algorithm proceeds with the next best query (w.r.t. the getScore function) until no further queries are available.

Algorithm 1 stops if the difference in the probabilities of the top two diagnoses is greater than the acceptance threshold σ or if no query can be used to differentiate between the remaining diagnoses (i.e. the score of the minimum score query equals to the maximum score of the used measure). The most probable diagnosis is then returned to the user. If it is impossible to differentiate between a number of highly probable minimal diagnoses, the algorithm returns a set that includes all of them. Moreover, in the first case (termination due to σ), the algorithm can continue if the user is not satisfied with the returned diagnosis and at least one further query exists.

Additional performance improvements can be achieved by using greedy strategies in Algorithm 2. The idea is to guide the search such that a leaf node of the left-most branch of a search tree contains a set of diagnoses $\mathbf{D^P}$ that might result in a tuple $\left\langle Q, \mathbf{D^P}, \mathbf{D^N}, \mathbf{D^\emptyset}\right\rangle$ with a low-score query. This method is based on the property of Equation 3 that $sc(Q) = 0$ if

$$\sum_{\mathcal{D}_i \in \mathbf{D^P}} p(\mathcal{D}_i) = \sum_{\mathcal{D}_j \in \mathbf{D^N}} p(\mathcal{D}_j) = 0.5 \quad \text{and} \quad p(\mathbf{D^\emptyset}) = 0$$

Consequently, the query selection problem can be presented as a two-way number partitioning problem: given a set of numbers, divide them into two sets such that the difference between the sums of the numbers in each set is as small as possible. The Complete Karmarkar-Karp (CKK) algorithm [20], which is one of the best algorithms developed for the two-way partitioning problem, corresponds to an extension of the Algorithm 2 with a set differencing heuristic [21]. The algorithm stops if the optimal solution to the two-way partitioning problem is found or if there are no further subsets to be investigated. In the latter case the best found solution is returned.

The main drawback of applying CKK to the query selection process is that none of the pruning techniques can be used. Also even if the algorithm finds an optimal solution to the two-way partitioning problem there just might be no query for a found set of diagnoses $\mathbf{D^P}$. Moreover, since the algorithm is complete it still has to investigate all subsets of the set of diagnoses in order to find the minimum score query. To avoid this exhaustive search we extended CKK with an additional termination criterion: the search stops if a query is found with a score below some predefined threshold γ. In our evaluation section we demonstrate substantial savings by applying the CKK partitioning algorithm.

To sum up, the proposed method depends on the efficiency of the classification/realization system and consistency/coherency checks given a particular ontology. The number of calls to a reasoning system can be reduced by decreasing the number of leading diagnoses n. However, the more leading diagnoses provide the more data for generating the next best query. Consequently, by varying the number of leading diagnoses it is possible to balance runtime with the number of queries needed to isolate the target diagnosis.[5]

5. Evaluation

We evaluated our approach using the real-world ontologies presented in Table 8 with the aim of demonstrating its applicability real-world settings. In addition, we employed generated examples to perform controlled experiments where the number of minimal diagnoses and their cardinality could be varied to make the identification of the target diagnosis more difficult. Finally, we carried out a set of tests using randomly modified large real-world ontologies to provide some insights on the scalability of the suggested debugging method.

For the first test we created a generator which takes a consistent and coherent ontology, a set of fault patterns together with their probabilities, the minimum number of minimum cardinality diagnoses m, and the required cardinality $|\mathcal{D}_t|$ of these minimum cardinality diagnoses as inputs. We also assumed that the target diagnosis has cardinality $|\mathcal{D}_t|$. The output of the generator is an alteration of the input ontology for which at least the given number of minimum cardinality diagnoses with the required cardinality exist. Furthermore, to introduce inconsistencies (incoherencies), the generator applies fault patterns randomly to the input ontology depending on their probabilities.

In this experiment we took five fault patterns from a case study reported by Rector et al. [5] and assigned fault probabilities according to their observations of typical user errors. Thus we assumed that in cases (a) and (b) (see Section 2.3), where an axiom includes some roles (i.e. property assertions), axiom descriptions are faulty with a probability of 0.025, in cases (c) and (d) 0.01 and in case (e) 0.001. In each iteration, the generator randomly selected an axiom to be altered and applied a fault pattern. Following this, another axiom was selected using the concept taxonomy and altered correspondingly to introduce an inconsistency (incoherency). The fault patterns were randomly selected in each step using the probabilities provided above.

For instance, given the description of a randomly selected concept A and the fault pattern "misuse of negation", we added the construct $\sqcap\neg X$ to the description of A, where X is a new concept name. Next, we randomly selected concepts B and S such that $S \sqsubseteq A$ and $S \sqsubseteq B$ and added $\sqcap X$ to the description of B. During the generation process, we applied the HS-Tree algorithm after each introduction of an incoherency/inconsistency to control two parameters: the minimum number of minimal cardinality diagnoses in the ontology and their cardinality. The generator continues to introduce incoherences/inconsistencies until the specified parameter values are reached. For instance, if the minimum number of minimum cardinality diagnoses is equal to $m = 6$ and their cardinality is $|\mathcal{D}_t| = 4$, then the generated ontology will include at least 6 diagnoses of cardinality 4 and possibly some additional number of minimal diagnoses of higher cardinalities.

The resulting faulty ontology as well as the fault patterns and their probabilities were inputs for the ontology debugger. The acceptance threshold σ was set to 0.95 and the number of most probable minimal diagnoses n was set to 9. In addition, one of the minimal diagnoses with the required cardinality was randomly selected as the target diagnosis. Note, the target ontology is not equal to the original ontology, but rather a corrected version of the altered one in which the faulty axioms were repaired by replacing them with their original (correct) versions according to the target diagnosis. The tests were performed using the ontologies bike2 to bike9, bcs3, galen and galen2 from Racer's benchmark suite[6].

[5] The source code as well as precompiled binaries can be downloaded from `http://rmbd.googlecode.com`. The package also includes a Protégé-plugin implementing the methods as described.

[6] Available at `http://www.racer-systems.com/products/download/benchmark.phtml`

	Ontology	DL	Axioms	#C/#P/#I	#CS/min/max	#D/min/max	Domain
1.	Chemical	$\mathcal{ALCHF}^{(D)}$	144	48/20/0	6/5/6	6/1/3	Chemical elements
2.	Koala	$\mathcal{ALCON}^{(D)}$	44	21/5/6	3/4/4	10/1/3	Training
3.	Sweet-JPL	$\mathcal{ALCHOF}^{(D)}$	2579	1537/121/50	1/13/13	13/1/1	Earthscience
4.	miniTambis	$\mathcal{ALCN}$	173	183/44/0	3/2/6	48/3/3	Biological science
5.	University	$\mathcal{SOIN}^{(D)}$	49	30/12/4	4/3/5	90/3/4	Training
6.	Economy	$\mathcal{ALCH}^{(D)}$	1781	339/53/482	8/3/4	864/4/8	Mid-level
7.	Transportation	$\mathcal{ALCH}^{(D)}$	1300	445/93/183	9/2/6	1782/6/9	Mid-level

Table 8: Diagnosis results for several of the real-world ontologies presented in [8]. #C/#P/#I are the number of concepts, properties and individuals in each ontology. #CS/min/max are the number of conflict sets, and their minimum and maximum cardinality. The same notation is used for diagnoses #D/min/max. The ontologies are available upon request.

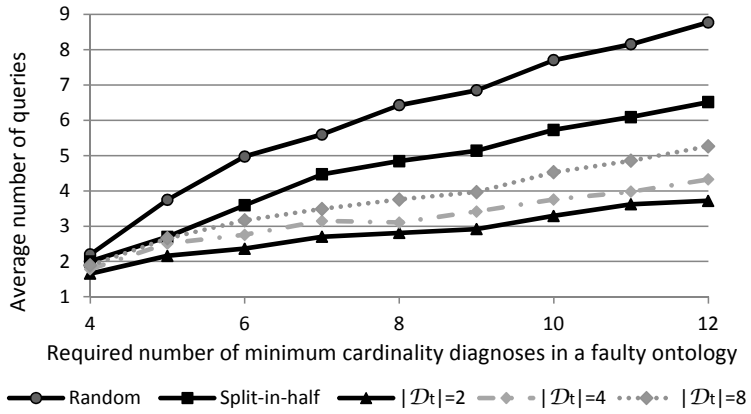

Figure 2: Average number of queries required to select the target diagnosis $\mathcal{D}_t$ with threshold $\sigma = 0.95$. Random and "split-in-half" are shown for the cardinality of minimal diagnoses $|\mathcal{D}_t| = 2$.

The average results of the evaluation performed on each test ontology (presented in Figure 2) show that the entropy-based approach outperforms the "split-in-half" heuristic as well as the random query selection strategy by more than 50% for the $|\mathcal{D}_t| = 2$ case due to its ability to estimate the probabilities of diagnoses and to stop once the target diagnosis crossed the acceptance threshold. On average the algorithm required 8 seconds to generate a query. In addition, Figure 2 shows that the number of queries required increases as the cardinality of the target diagnosis increases, regardless of the method. Despite this, the entropy-based approach remains better than the "split-in-half" method for diagnoses with increasing cardinality. The approach did however require more queries to discriminate between high cardinality diagnoses because in such cases more minimal conflicts were generated. Consequently, the debugger should consider more minimal diagnoses in order to identify the target one.

For the next test we selected seven real-world ontologies described in Tables 8 and 9[7]. Performance of both the entropy-based and "split-in-half" selection strategies was evaluated using a variety of different prior fault probabilities to investigate under which conditions the entropy-based method should be preferred.

In our experiments we distinguished between three different distributions of prior fault probabilities: extreme, moderate and uniform (see Figure 3 for an example). The *extreme distribution* simulates a situation in which very high failure probabilities are assigned to a small number of syntax elements. That is, the provider of the estimates is quite sure that exactly these elements are causing a fault. For instance, it may be well known that a user has problems formulating restrictions in OWL whereas all other elements, such as subsumption and conjunction, are well understood. In the case of a *moderate distribution* the estimates provide a slight bias towards some syntax elements. This distribution has the same motivation as the extreme one, however,

[7]All experiments were performed on a PC with Core2 Duo (E8400), 3 Ghz with 8 Gb RAM, running Windows 7 and Java 6.

		Leading diagnoses			All diagnoses		
Ontology		Consistency	Conflicts	Diagnoses	Consistency	Conflicts	Diagnoses
Chemical	time	0/3/8	90/107/128	1/97/326	0/3/18	105/130/179	2/126/402
	calls	264	6	7	262	6	7
		runtime: 723			runtime: 892		
Koala	time	0/1/3	19/25/30	0/11/70	0/2/4	24/30/37	0/12/105
	calls	74	3	10	75	3	11
		runtime: 120			runtime: 148		
Sweet-JPL	time	1/31/112	5185/5185/5185	0/586/5332	31/106/195	5192/5192/5192	1/438/5319
	calls	187	1	10	195	1	14
		runtime: 5991			runtime: 6312		
miniTambis	time	0/5/14	84/157/210	0/57/504	1/5/15	88/167/225	3/19/537
	calls	111	3	10	189	3	49
		runtime: 586			runtime: 1027		
University	time	0/2/3	31/41/54	0/20/157	0/2/5	37/46/60	2/5/200
	calls	126	4	10	283	4	91
		runtime: 205			runtime: 536		
Economy	time	1/12/26	410/460/569	0/282/2085	1/9/80	418/510/681	16/25/1929
	calls	239	6	10	2064	8	865
		runtime: 2857			runtime: 25369		
Transportaton	time	0/11/58	237/438/683	0/352/3176	1/9/130	222/429/636	16/29/6394
	calls	337	7	10	3966	9	1783
		runtime: 3671			runtime: 65010		

Table 9: Min/avg/max time and calls required to compute the nine leading most probable diagnoses as well as all diagnoses for the real-world ontologies. Values are given for each stage, i.e. consistency checking, computation of minimal conflicts and minimal diagnoses, together with the total runtime needed to compute the diagnoses. All time values are 15 trial averages and are given in milliseconds.

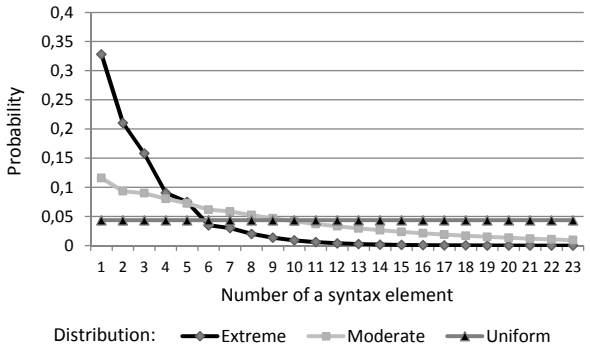

Figure 3: Example of prior fault probabilities of syntax elements sampled from extreme, moderate and uniform distributions.

in this case the probability estimator is less sure about the sources of possible errors in axioms. Both extreme and moderate distributions correspond to the exponential distribution with $\lambda = 1.75$ and $\lambda = 0.5$ respectively. The *uniform distribution* models the situation where no prior fault probabilities are provided and the system assigns equal probabilities to all syntax elements found in a faulty ontology. Of course the prior probabilities of diagnoses may not reflect the actual situation. Therefore, for each of the three distributions we differentiate between good, average and bad cases. In the *good case* the estimates of the prior fault probabilities are correct and the target diagnosis is assigned a high probability. The *average case* corresponds to the situation when the target diagnosis is neither favored nor penalized by the priors. In the *bad case* the prior distribution is unreasonable and disfavors the target diagnosis by assigning it a low probability.

We executed 30 tests for each of the combinations of the distributions and cases with an acceptance threshold $\sigma = 0.85$ and a required number of most probable minimal diagnoses $n = 9$. Each iteration started with the generation of a set of prior fault probabilities of syntax elements by sampling from a selected distribution (extreme, moderate or uniform). Given the priors we computed the set of all minimal diagnoses $\mathbf{D}$ of a given ontology and selected the target one according to the chosen case (good, average or bad). In the good case the prior probabilities favor the target diagnosis and, therefore, it should be selected from the diagnoses with high probability. The set of diagnoses was ordered according

to their probabilities and the algorithm iterated through the set starting from the most probable element. In the first iteration the most probable minimal diagnosis $\mathcal{D}_1$ is added to the set G. In next iteration j a diagnosis $\mathcal{D}_j$ was added to the set G if $\sum_{i \leq j} p(\mathcal{D}_i) \leq \frac{1}{3}$ and to the set A if $\sum_{i \leq j} p(\mathcal{D}_i) \leq \frac{2}{3}$. The obtained set G contained all most probable diagnoses which we considered as good. All diagnoses in the set $A \setminus G$ were classified as average and the remaining diagnoses $\mathbf{D} \setminus A$ as bad. Depending on the selected case we randomly selected one of the diagnoses as the target from the appropriate set.

The results of the evaluation presented in Table 10 show that the entropy-based query selection approach clearly outperforms "split-in-half" in good and average cases for the three probability distributions. The average time required by the debugger to perform such basic operations as consistency checking, computation of minimal conflicts and diagnoses is presented in Table 11. The results indicate that on average at most 17 seconds required to compute up to 9 minimal diagnoses and a query. Moreover, the number of axioms in a query remains reasonable in most of the cases stays bounds, i.e. between 1 and 4 axioms per query.

In the uniform case better results were observed since the diagnoses have different cardinality and structure, i.e. they include different syntax elements. Consequently, even if equal probabilities for all syntax elements (uniform distribution) are given, the probabilities of diagnoses are different. Axioms with a greater number of syntax elements receive a higher fault probability. Also, diagnoses with a smaller cardinality in many cases receive a higher probability. This information provides enough bias to favor the entropy-based method.

In the bad case, where the target diagnosis received a low probability and no information regarding the prior fault probabilities was given, we observed that the performance of the entropy-method improved as more queries were posed. In particular, in the University ontology the performance is essentially similar (7.27 vs. 7.37) whereas in the Economy and Transportation ontology the entropy-based method can save and average of two queries.

"Split-in-half" appears to be particularly inefficient in all good, average and bad cases when applied to ontologies with a large number of minimal diagnoses, such as Economy and Transportation. The main problem is that no stop criteria can be used with the greedy method as it is unable to provide any ordering on the set of diagnoses. Instead, the method continues until no further queries can be generated, i.e. only one minimal diagnosis exists or there are no discriminating queries. Conversely, the entropy-based method is able to improve its probability estimates using Bayes-updates as more queries are answered and to exploit the differences in the probabilities in order to decide when to stop.

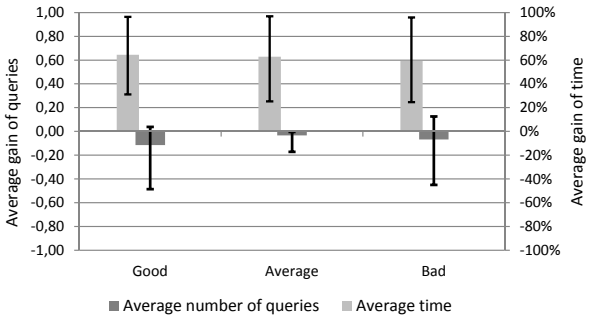

Figure 4: Average time/query gain resulting from the application of the extended CKK partitioning algorithm. The whiskers indicate the maximum and minimum possible average gain of queries/time using extended CKK.

The most significant gains are achieved for ontologies with many minimal diagnoses and for the average and good cases, e.g. the target diagnosis is within the first or second third of the minimal diagnoses ranked by their prior probability. In these cases the entropy-based method can save up to 60% of the queries.

Therefore, we can conclude that even rough estimates of the prior fault probabilities are sufficient, provided that the target diagnosis is not significantly penalized. Even if no fault probabilities are available and there are many minimal diagnoses, the entropy-based method is advantageous. The differences between probabilities of individual syntax elements appears not to influence the results of the query selection process and affect only the number of outliers, i.e. cases in which the diagnosis approach required either few or many queries compared to the average.

Another interesting observation is that often both methods eliminated more than n diagnoses in one iteration. For instance, in the case of the Transportation ontology both methods were able to remove hundreds of minimal diagnoses with a small number of queries. This behavior appears to stem from relations between the diagnoses. That is, the addition of a query to either P or N allows the method to remove not only the diagnoses in sets $\mathbf{D^P}$ or $\mathbf{D^N}$, but also some unobserved diagnoses that were not in any of the sets of n leading diagnoses computed by HS-Tree. Given the sets P and N, HS-Tree automatically invalidates all diagnoses which do not fulfill the requirements (see Definition 2).

The extended CKK method presented in Section 4 was evaluated in the same settings as the complete Al-

Entropy-based query selection

Ontology	Case	Distribution								
		Extreme			Moderate			Uniform		
		min	avg	max	min	avg	max	min	avg	max
Chemical	Good	1	**1.63**	3	1	**1.7**	2	1	**1.83**	2
	Avg.	1	**1.87**	4	1	**1.73**	3	1	**1.7**	2
	Bad	2	3.03	4	2	3.03	4	2	3.17	4
Koala	Good	1	**1.7**	3	1	**2.4**	4	1	**2.67**	3
	Avg.	1	**1.8**	3	1	**2.37**	4	1	**2.4**	3
	Bad	1	3.5	6	2	4.33	7	3	4.13	5
Sweet-JPL	Good	1	**3.27**	7	2	**3.43**	7	3	**3.87**	7
	Avg.	1	**3.5**	6	1	4.03	7	3	4.07	6
	Bad	3	3.93	6	2	4.03	6	3	**3.37**	4
miniTambis	Good	1	**2.37**	4	2	**2.73**	4	2	**2.77**	3
	Avg.	1	**2.53**	4	2	**4.03**	8	3	**4.53**	7
	Bad	3	6.43	11	3	7.93	17	5	9.03	13
University	Good	1	**2.7**	4	3	**3.83**	7	3	**4.4**	8
	Avg.	1	**3.4**	6	3	7.03	12	4	**7.27**	10
	Bad	5	9.13	15	5	9.7	14	6	10.03	14
Economy	Good	1	**3.2**	11	3	**3.1**	4	3	**3.93**	6
	Avg.	1	**4.63**	14	3	**5.57**	12	5	**6.5**	8
	Bad	8	12.3	19	6	11.5	21	7	11.67	19
Transportation	Good	1	**5.63**	14	1	**6.97**	12	3	**9.5**	14
	Avg.	1	**6.9**	16	1	**7.73**	12	3	**8.73**	14
	Bad	3	**12.4**	18	8	**12.8**	20	3	**12.1**	18

"Split-in-half" query selection

Chemical	Good	2	2.63	3	2	2.7	3	2	2.53	3
	Avg.	2	2.63	3	2	2.67	3	2	2.77	3
	Bad	2	**2.63**	3	2	**2.6**	3	2	**2.4**	3
Koala	Good	3	3.3	4	3	3.3	4	3	3.47	4
	Avg.	3	3.33	4	3	3.2	4	3	3.23	4
	Bad	3	**3.43**	4	3	**3.4**	4	3	**3.5**	4
Sweet-JPL	Good	3	3.83	4	3	3.8	4	4	4	4
	Avg.	3	3.57	4	3	**3.8**	4	3	**3.47**	4
	Bad	3	**3.87**	4	3	**3.8**	4	3	3.8	4
miniTambis	Good	4	5.33	6	4	5	6	4	4	4
	Avg.	4	5.1	6	4	4.93	7	5	5.43	7
	Bad	5	**5.93**	8	4	**5.8**	7	5	**6.3**	7
University	Good	4	5.93	8	4	6	8	4	5.43	8
	Avg.	4	5.87	7	5	**6.73**	9	6	7.37	8
	Bad	5	**6.97**	9	5	**7.2**	9	5	**7**	8
Economy	Good	6	7.87	11	6	7.4	10	6	7.5	10
	Avg.	6	8	12	5	7.63	12	6	8.73	13
	Bad	9	**11.50**	14	6	**11.1**	14	8	**11.3**	15
Transportation	Good	5	8.03	13	5	7.3	11	6	11.43	18
	Avg.	3	9	16	5	9.4	13	5	11.43	18
	Bad	10	12.67	19	7	13	19	6	13.8	20

Table 10: Minimum, average and maximum number of queries required by the entropy-based and "split-in-half" query selection methods to identify the target diagnosis in real-world ontologies. Ontologies are ordered by the number of diagnoses.

Ontology	Good			Average			Bad		
	DT	QT	QL	DT	QT	QL	DT	QT	QL
Chemical	459.33	117.67	3	461.33	121	3.34	256.67	75.67	2.19
Koala	88.33	1308.33	3.47	92	1568.67	3.90	56.33	869.33	2.36
Sweet-JPL	2387.33	691.67	1.48	2272	926	1.61	2103	1240.33	1.57
miniTabmis	481.33	2764.33	3.27	398.33	2892	2.53	238.67	3223	1.76
University	189.33	822.67	3.91	145	903.33	2.82	113	872	2.11
Economy	2953.33	6927	3.06	3239	8789	3.80	3083	8424.67	1.58
Transportation	6577.33	9426.33	2.37	7080.67	10135.33	2.29	7186.67	9599.67	1.64

Table 11: Average time required to compute at most nine minimal diagnoses (DT) and a query (QT) in each iteration, as well as the average number of axioms in a query after minimization (QL). The averages are shown for extreme, moderate and uniform distributions using the entropy-based query selection method. Time is measured in milliseconds.

gorithm 2 with acceptance threshold $\gamma = 0.1$. The obtained results presented in Figure 4 show that the extended CKK method decreases the length of a debugging session by at least 60% while requiring on average 0.1 queries more than Algorithm 2. In some cases (mostly for the uniform distribution) the debugger using CKK search required even fewer queries than Algorithm 2 because of the inherent uncertainty of the domain. The plot of the average time required by Algorithm 2 and CKK to identify the target diagnosis presented in Figure 5 shows that the application of the latter can reduce runtime significantly.

In the last experiment we tried to simulate an expert developing large real-world ontologies[8] as described in Table 12. Often in such settings an expert makes small changes to the ontology and then runs the reasoner to verify that the changes are valid, i.e. the ontology is consistent and its entailments are correct. To simulate this scenario we used the generator described in the first experiment to introduce 1 to 3 random changes that would make the ontology incoherent. Then, for each modified ontology, we performed 15 tests using the fault distributions as in the second test. The results obtained by the entropy-based query selection method using CKK for query computation are presented in Table 13. These results show that the method can be used for analysis of large ontologies with over 33000 axioms while requiring a user to wait for only a minute to compute the next query.

6. Related work

Despite the range of ontology diagnosis methods available (see [7, 8, 9]), to the best of our knowledge no interactive ontology debugging methods, such as our "split-in-half" or entropy-based methods, have been proposed so far. The idea of ranking of diagnoses and proposing a target diagnosis is presented in [11]. This method uses a number of measures such as: (a) the frequency with which an axiom appears in conflict sets, (b) impact on an ontology in terms of its "lost" entailments when an axiom is modified or removed, (c) ranking of test cases, (d) provenance information about axioms, and (e) syntactic relevance. For each axiom in a conflict set, these measures are evaluated and combined to produce a rank value. These ranks are then used by a modified HS-Tree algorithm to identify diagnoses with a minimal rank. However, the method fails when a target diagnosis cannot be determined reliably with the given a-priori knowledge. In our work required information is acquired until the target diagnosis can be identified with confidence. In general, the work of [11] can be combined with the ideas presented in this paper as axiom ranks can be taken into account together with other observations for calculating the prior probabilities of the diagnoses.

The idea of selecting the next best query based on the expected entropy was exploited in the generation of decisions trees in [22] and further refined for selecting measurements in the model-based diagnosis of circuits in [18]. We extend these methods to query selection in the domain of ontology debugging.

In the area of debugging logic programs, Shapiro [23] developed debugging methods based on query answering. Roughly speaking, Shapiro's method aims to detect one fault at a time by querying an oracle about the intended behavior of a Prolog program at hand. In our terminology, for each answer that must not be entailed this diagnosis approach generates one conflict at a time by exploiting the proof tree of a Prolog program. The method then identifies a query that splits the conflict in half. Our approach can deal with multiple diagnoses and conflicts simultaneously which can be ex-

[8]The ontologies taken from TONES repository `http://owl.cs.manchester.ac.uk/repository`

Ontology	Cton	Opengalen-no-propchains
Axioms	33203	9664
DL	$\mathcal{SHF}$	$\mathcal{ALEHIF}^{(D)}$
#CS/min/max	6/3/7	9/5/8
#D/min/max	15/1/5	110/2/6
Consistency	5/209/1078	1/98/471
QuickXplain	17565/20312/38594	7634/10175/12622
Diagnosis	1/5285/38594	10/1043/19543
Overall runtime	146186	119973

Table 12: Statistics for the real-world ontologies used in the stress-tests measured for a single random alteration. #CS/min/max are the number of minimal conflict sets, and their minimum and maximum cardinality. The same notation is used for diagnoses #D/min/max. The minimum/average/maximum time required to make a consistency check (Consistency), compute a minimal conflict set (QuickXplain) and a minimal diagnosis are measured in milliseconds. Overall runtime indicates the time required to compute all minimal diagnoses in milliseconds.

Good					
Ontology	#Query	Overall	QT	DT	QL
Cton	3	176828	6918	52237	4
Opengalen-no-propchains	8	154145	2349	22905	4
Average					
Cton	4	177383	6583	52586	3
Opengalen-no-propchains	7	151048	3752	21344	4
Bad					
Cton	5	190407	5742	35608	1
Opengalen-no-propchains	14	177728	1991	11319	3

Table 13: Average values measured for extreme, moderate and uniform distributions in each of the good, average and bad cases. #Query is the number of queries required to find the target diagnosis. Overall runtime as well as the time required to compute a query (QT) and at least nine minimal diagnoses (DT) are given in milliseconds. Query length (QL) shows the average number of axioms in a query.

ploited by query generation strategies such as "split-in-half" and entropy-based methods. Whereas the "split-in-half" strategy splits the set of diagnoses in half, Shapiros's method focuses on one conflict. Furthermore, the exploitation of failure probabilities is not considered in [23]. However, Shapiro's method includes the learning of new clauses in order to cover not entailed answers. Interleaving discrimination of diagnoses and learning of descriptions is currently not considered in our approach because of their additional computational costs.

From a general point of view Shapiro's method can be seen as a prominent example of inductive logic programming (ILP) including systems such as [24, 25]. In particular, [25] proposes inverse entailments combined with general to specific search through a refinement graph with the goal of generating a theory (hypothesis) which covers the examples and fulfills additional properties. Compared to ILP, the focus of our work lies on the theory revision. However, our knowledge representation languages are variants of description logics and not logic programs. Moreover, our method aims to discover axioms which must be changed while minimizing user interaction. Preferences of theory changes are expressed by probabilities which are updated through Bayes' rule. Other preferences based on plausible extensions of the theory were not considered, again because of their computational costs.

Although model-based diagnosis has also been applied to logic programs [26], constraint knowledge-bases [27] and hardware descriptions [28], none of these approaches propose a query generation method to discriminate between diagnoses.

7. Conclusions

In this paper we presented an approach to the interactive diagnosis of ontologies. This approach is applicable to any ontology language with monotonic semantics. We showed that the axioms generated by classification and realization reasoning services can be exploited to generate queries which differentiate between diagnoses. For selecting the best next query we proposed two strategies: The "split-in-half" strategy prefers

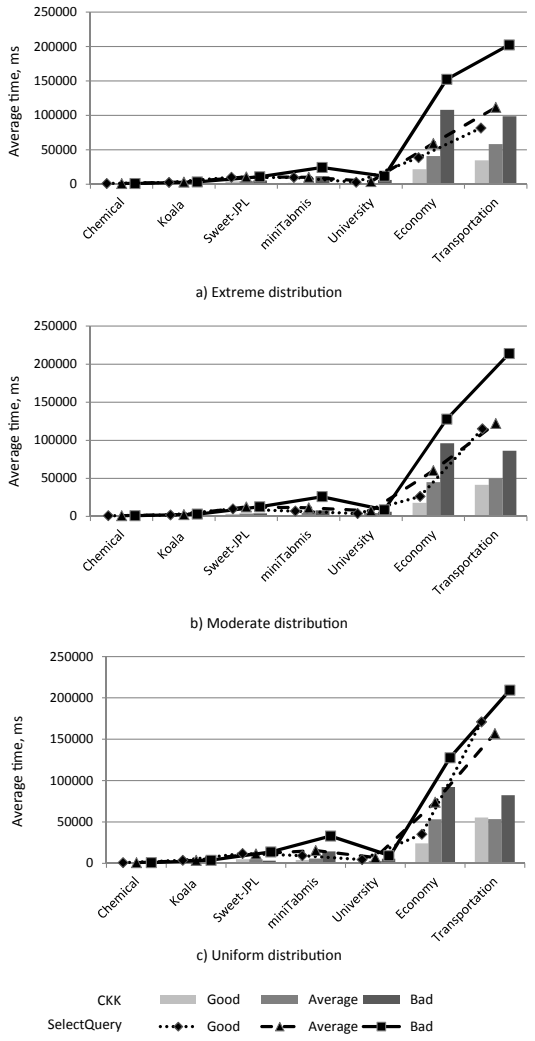

Figure 5: Average time required to identify the target diagnosis using CKK and brute force query selection algorithms.

queries which allow eliminating a half of leading diagnoses. The entropy-based strategy employs information theoretic concepts to exploit knowledge about the likelihood of axioms needing to be changed because the ontology at hand is faulty. Based on the probability of an axiom containing an error we predict the information gain produced by a query result, enabling us to select the best subsequent query according to a one-step-lookahead entropy-based scoring function. We described the implementation of a interactive debugging algorithm and compared the entropy-based method with the "split-in-half" strategy. Our experiments showed a significant reduction in the number of queries required to identify the target diagnosis when the entropy-based method is applied. Depending on the quality of the prior probabilities the number of queries required may be reduced by up to 60%.

In order to evaluate the robustness of the entropy-based method we experimented with different prior fault probability distributions as well as different qualities of the prior probabilities. Furthermore, we investigated cases where knowledge about failure probabilities is missing or inaccurate. Where such knowledge is unavailable, the entropy-based methods ranks the diagnoses based on the number of syntax elements contained in an axiom and the number of axioms in a diagnosis. If we assume that this is a reasonable guess (i.e. the target diagnosis is not at the lower end of the diagnoses ranked by their prior probabilities) then the entropy-based method outperforms "split-in-half". Moreover, even if the initial guess is not reasonable, the entropy-based method improves the accuracy of the probabilities as more questions are asked. Furthermore, the applicability of the approach to real-world ontologies containing thousand of axioms was demonstrated by extensive set of evaluations which are publicly available.

[1] K. Shchekotykhin, G. Friedrich, Query strategy for sequential ontology debugging, in: P. F. Patel-Schneider, P. Yue, P. Hitzler, P. Mika, Z. Lei, J. Pan, I. Horrocks, B. Glimm (Eds.), The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, Shanghai, China, 2010, pp. 696–712.

[2] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: The next step for OWL, Web Semantics: Science, Services and Agents on the World Wide Web 6 (4) (2008) 309–322.

[3] J. Ceraso, A. Provitera, Sources of error in syllogistic reasoning, Cognitive Psychology 2 (4) (1971) 400–410.

[4] P. N. Johnson-Laird, Deductive reasoning, Annual review of psychology 50 (1999) 109–135.

[5] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, C. Wroe, OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns, in: E. Motta, N. R. Shadbolt, A. Stutt, N. Gibbins (Eds.), Engineering Knowledge in the Age of the SemanticWeb 14th International Conference, EKAW 2004, Springer, Whittenbury Hall, UK, 2004, pp. 63–81.

[6] C. Roussey, O. Corcho, L. M. Vilches-Blázquez, A catalogue of OWL ontology antipatterns, in: International Conference On Knowledge Capture, ACM, Redondo Beach, California, USA, 2009, pp. 205–206.

[7] S. Schlobach, Z. Huang, R. Cornet, F. Harmelen, Debugging Incoherent Terminologies, Journal of Automated Reasoning 39 (3) (2007) 317–349.

[8] A. Kalyanpur, B. Parsia, M. Horridge, E. Sirin, Finding all Justifications of OWL DL Entailments, in: K. Aberer, K.-S. Choi, N. F. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.), The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Vol. 4825 of LNCS, Springer Verlag, Berlin, Heidelberg, 2007, pp. 267–280.

[9] G. Friedrich, K. Shchekotykhin, A General Diagnosis Method for Ontologies, in: Y. Gil, E. Motta, R. Benjamins, M. Musen (Eds.), The Semantic Web - ISWC 2005, 4th International Se-

mantic Web Conference, Springer, 2005, pp. 232–246.
[10] M. Horridge, B. Parsia, U. Sattler, Laconic and Precise Justifications in OWL, Proc of the 7th International Semantic Web Conference ISWC 2008 5318 (2008) 323–338.
[11] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca-Grau, Repairing Unsatisfiable Concepts in OWL Ontologies, in: Y. Sure, J. Domingue (Eds.), The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Vol. 4011 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 170–184.
[12] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Web Semantics: Science, Services and Agents on the World Wide Web 5 (2) (2007) 51–53.
[13] V. Haarslev, R. Müller, RACER System Description, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), 1st International Joint Conference on Automated Reasoning, Vol. 2083 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 701–705.
[14] B. Motik, R. Shearer, I. Horrocks, Hypertableau Reasoning for Description Logics, Journal of Artificial Intelligence Research 36 (1) (2009) 165–228.
[15] A. Borgida, On the relative expressiveness of description logics and predicate logics, Artificial Intelligence 82 (1-2) (1996) 353–367.
[16] F. Baader, Appendix: Description Logic Terminology, in: P. F. Baader, Franz and Calvanese, Diego and McGuinness, Deborah L. and Nardi, Daniele and Patel-Schneider (Ed.), Description Logic Handbook, Cambridge University Press, 2003, pp. 485–495.
[17] R. Reiter, A Theory of Diagnosis from First Principles, Artificial Intelligence 32 (1) (1987) 57–95.
[18] J. de Kleer, B. C. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1) (1987) 97–130.
[19] U. Junker, QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems, in: D. L. McGuinness, G. Ferguson (Eds.), Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, Vol. 3, AAAI Press / The MIT Press, 2004, pp. 167–172.
[20] R. E. Korf, A complete anytime algorithm for number partitioning, Artificial Intelligence 106 (2) (1998) 181–203.
[21] N. Karmarkar, R. M. Karp, G. S. Lueker, A. M. Odlyzko, Probabilistic analysis of optimum partitioning, Journal of Applied Probability 23 (3) (1986) 626–645.
[22] J. R. Quinlan, Induction of Decision Trees, Machine Learning 1 (1) (1986) 81–106.
[23] E. Y. Shapiro, Algorithmic Program Debugging, MIT Press, 1983.
[24] S. Muggleton, W. L. Buntine, Machine Invention of First-order Predicates by Inverting Resolution, in: J. Laird (Ed.), Proceedings of the 5th International Conference on Machine Learning ({ICML'88}), Morgan Kaufmann, 1988, pp. 339–352.
[25] S. Muggleton, Inverse Entailment and Progol, New Generation Computing, Special issue on Inductive Logic Programming 13 (3-4) (1995) 245–286.
[26] L. Console, G. Friedrich, D. T. Dupre, Model-Based Diagnosis Meets Error Diagnosis in Logic Programs, in: IJCAI, 1993, pp. 1494–1501.
[27] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, Consistency-based diagnosis of configuration knowledge bases, Artificial Intelligence 152 (2004) 213–234.
[28] G. Friedrich, M. Stumptner, F. Wotawa, Model-based diagnosis of hardware designs, Artif. Intell. 111 (1-2) (1999) 3–39.