Balancing Brave and Cautious Query Strategies in Ontology Debugging

Patrick Rodler, Kostyantyn Shchekotykhin, Philipp Fleiss and Gerhard Friedrich *

Alpen-Adria-Universität Klagenfurt
Universitätsstrasse 65-67
9020 Klagenfurt, Austria
firstname.lastname@aau.at

Abstract. Sequential ontology debugging is aimed at the efficient discrimination between diagnoses, i.e. sets of axioms which must be altered or deleted from the ontology to restore consistency. By querying additional information the number of possible diagnoses can be gradually reduced. The selection of the best queries is crucial for minimizing diagnosis costs. If prior fault probabilities (FPs) are available, the best results are achieved by entropy based query selection. Given that FPs are only weakly justified, however, this strategy bravely suggests suboptimal queries although more cautious strategies should be followed. In such a case, it is more efficient to follow a no-risk strategy which prefers queries that eliminate 50% of diagnoses independently of any FPs. However, choosing the appropriate strategy in advance is impossible because the quality of given priors cannot be assessed before additional information is queried. We propose a method which combines advantages of both approaches. On the one hand, the method takes into account available meta information in terms of FPs and the user's confidence in these. On the other hand, the method can cope with weakly justified FPs by limiting the risk of suboptimal query selections based on the user's confidence in the FPs. The readiness to take risk is adapted depending on the outcome of previous queries. Our comprehensive evaluation shows that the proposed debugging method significantly reduces the number of queries compared to both the entropy based and the no-risk strategy for any choice of FPs.

1 Introduction

Support of ontology development and maintenance is an important requirement for the extensive use of Semantic Web technologies. However, the correct formulation of logical descriptions is an error-prone task even for experienced knowledge engineers. Ontology debuggers [8, 3, 1] assist ontology development by identifying sets of axioms (called diagnoses) that have to be modified s.t. inconsistencies or unwanted entailments are avoided. In many cases the main problem of debugging is the big number of alternative diagnoses.

To solve the problem a set of heuristics to rank diagnoses was proposed by [4]. However, this solution is not satisfactory as it cannot be guaranteed that the top-ranked

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

diagnosis is the target diagnosis and no further assistance for diagnoses discrimination is provided. Therefore, a debugging method based on active learning was proposed by [9]. The approach exploits the fact that an ontology without the axioms of one diagnosis may have different entailments than the ontology without the axioms of another diagnosis. Using these entailments the algorithm queries the user or another oracle whether a set of axioms is entailed by the target ontology or not. The answers are used to eliminate disagreeing diagnoses. The algorithm continues to query the oracle until a diagnosis (the target diagnosis) is significantly more probable than any other. To minimize the number of queries the algorithm uses some meta information, namely the fact that some errors are more common than others [6], i.e. that different language constructs have different fault probabilities. These probabilities can either be extracted from the logs of previous sessions or specified by the user expressing own beliefs. This advantage might be lost if unsuitable probabilities are provided where the target diagnosis is rather improbable. In this case the entropy-based query selection as well as active learning methods might require significantly more queries than strategies like split-in-half which behave independently of any meta information.

We present a hybrid method that outperforms both types of strategies by combining their benefits while overcoming their drawbacks. On the one hand, our method takes advantage of the given meta information as long as good performance is achieved. On the other hand, it gradually gets more independent of meta information if suboptimal behavior is measured. Moreover, our strategy takes into account a user's subjective quality estimation of the meta information. In this way a user may decide to take influence on the algorithm's behavior or let the algorithm act freely and find the best strategy on its own. To find the best strategy, our method constantly improves the quality of meta information as well as it learns to act profitably in all kinds of situations by exploiting its adaptiveness. In our comprehensive evaluation we demonstrate for various types and quality of meta information that our approach is very robust and that query costs in comparison to existing solutions are substantially minimized in all situations.

The motivation of our work as well as basic concepts are provided in Section 2. Section 3 gives the details of the suggested approach. Implementation details are discussed in Section 4 and evaluation results are described in Section 5.

2 Basic Concepts and Motivation

Ontology Debugging deals with the following problem:

Problem Definition 1 (Ontology Debugging) ***Given*** *a diagnosis problem instance* $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$ *where* $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ *denotes an ontology, i.e. a set of terminological axioms* $\mathcal{T}$ *and a set of assertional axioms* $\mathcal{A}$*,* $\mathcal{B}$ *denotes a set of background knowledge axioms (which are assumed to be true),* TC^+ *is a set of positive test cases, and* TC^- *is a set of negative test cases, such that*

- $\mathcal{O}$ *is inconsistent/incoherent*[1] *and/or*

[1] Note that throughout the rest of this paper we consider debugging of inconsistent ontologies. However, the same approach can also be used to restore coherency.

- $\exists\, t^+ \in TC^+ :\ \mathcal{O} \cup \mathcal{B} \not\models t^+$ *and/or*
- $\exists\, t^- \in TC^- :\ \mathcal{O} \cup \mathcal{B} \models t^-$.

The reasonable assumption is made that the background theory is consistent with the positive and negative test cases, i.e. $\mathcal{B} \cup (\bigwedge_{t^+ \in TC^+} t^+) \cup \neg t^-$ *is consistent for all* $t^- \in TC^-$*. The task is then to* ***find*** *a diagnosis* $\mathcal{D} \subseteq \mathcal{O}$ *s.t. there is a set of axioms* EX *s.t. 1 and 2 and 3 hold for* $\mathcal{O}^* := (\mathcal{O} \setminus \mathcal{D}) \cup EX$*:*

1. $\mathcal{O}^* \cup \mathcal{B}$ *is consistent/coherent*
2. $\mathcal{O}^* \cup \mathcal{B} \models t^+ \quad \forall t^+ \in TC^+$
3. $\mathcal{O}^* \cup \mathcal{B} \not\models t^- \quad \forall t^- \in TC^-$

$\mathcal{O}^*$ *denotes the target ontology and* EX *is called extension. A diagnosis* $\mathcal{D}$ *is minimal iff there is no* $\mathcal{D}' \subset \mathcal{D}$ *s.t.* $\mathcal{D}'$ *is a diagnosis. A diagnosis* $\mathcal{D}$ *gives complete information about the correctness of each axiom* $ax_k \in \mathcal{O}$*, i.e. all* $ax_i \in \mathcal{D}$ *are assumed to be faulty and all* $ax_j \in \mathcal{O} \setminus \mathcal{D}$ *are assumed to be correct.*

In ontology debugging two types of operations can be distinguished: diagnosis and repair. At first, one needs to find a set of axioms, i.e. a diagnosis $\mathcal{D}$, which must be deleted from the ontology $\mathcal{O}$ in order to restore consistency (requirements 1 and 3). This is the task of ontology diagnosis. Then it might be necessary to add appropriate axioms EX to $\mathcal{O} \setminus \mathcal{D}$ in order to give the ontology the intended semantics/entailments (requirement 2) because by deleting axioms from $\mathcal{O}$ some intended entailments might be eliminated as well. This is the topic of ontology repair. In this work we focus on ontology diagnosis. For EX we use the following approximation: $EX \approx \bigwedge_{t^+ \in TC^+} t^+$. Note that EX must at least contain all axioms in TC^+.

Ontology diagnosis can be exemplified by the following simple OWL DL ontology $\mathcal{O}_{ex} = \mathcal{T} \cup \mathcal{A}$ with terminology $\mathcal{T}$:

$$ax_1 : A \sqsubseteq \forall s.B \qquad ax_2 : B \sqsubseteq \neg\exists v.\neg C \qquad ax_3 : C \sqsubseteq D \sqcap \neg D_1$$
$$ax_4 : D \sqsubseteq E \sqcap E_1 \qquad ax_5 : E \sqsubseteq F \qquad ax_6 : F \sqsubseteq R$$

and the assertions $\mathcal{A} : \{A(w), s(w,w), v(w,w), \neg R(w)\}$. Let us assume that all assertions are added to the background theory, i.e. $\mathcal{B} = \mathcal{A}$, and both sets TC^+ and TC^- are empty. Then the ontology $\mathcal{O}_{ex} = \mathcal{T}$ is inconsistent and the set of minimal diagnoses for this diagnosis problem instance $\langle \mathcal{T}, \mathcal{A}, \emptyset, \emptyset \rangle$ is $\mathbf{D} = \{\mathcal{D}_1 : [ax_1], \mathcal{D}_2 : [ax_2], \mathcal{D}_3 : [ax_3], \mathcal{D}_4 : [ax_4], \mathcal{D}_5 : [ax_5], \mathcal{D}_6 : [ax_6]\}$.

An algorithm for generating all possible diagnoses for a given diagnosis problem instance is presented in [1]. However, one major issue in ontology diagnosis is that there may be a huge number n of possible diagnoses $\mathbf{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, depending on the size and the properties of the ontology $\mathcal{O}$. This problem is also manifested in the above example where six possible diagnoses are found for an ontology consisting of merely six axioms. To tackle this problem, the authors in [9] exploit the fact that ontologies $\mathcal{O} \setminus \mathcal{D}_i$ and $\mathcal{O} \setminus \mathcal{D}_j$ have different entailments for $\mathcal{D}_i, \mathcal{D}_j \in \mathbf{D}$ $(\mathcal{D}_i \neq \mathcal{D}_j)$ in order to discriminate between potential diagnoses $\mathbf{D}$. They discuss methods which gather new information by posing queries about intended entailments of the target ontology to an oracle and utilize this information to reduce the set of potential diagnoses as quickly as possible. So, the following subproblem of ontology diagnosis is addressed:

Algorithm 1: Query Generation

Input: diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$, set of diagnoses $\mathbf{D}$, a reasoner R_E instantiated to provide all entailments of types $E \subseteq ET$
Output: a set of queries $\mathbf{X}$

```
foreach D^X ⊂ D do
    X ← getEntailments(R_E, B ∪ TC+ ∪ ⋂_{D∈D^X} O \ D);
    if X ≠ ∅ then
        foreach D_r ∈ D \ D^X do
            if O \ D_r ∪ B ∪ TC+ ⊨ X then D^X ← D^X ∪ {D_r};
            else if O \ D_r ∪ B ∪ TC+ ⊨ ¬X then D^¬X ← D^¬X ∪ {D_r};
            else D^∅ ← D^∅ ∪ {D_r};
        X ← X ∪ {(X, ⟨D^X, D^¬X, D^∅⟩)}
return X;
```

Problem Definition 2 (Diagnosis Discrimination) ***Given*** *the set of possible diagnoses* $\mathbf{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ *w.r.t.* $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$, ***find*** *a sequence* $(X_1, \ldots, X_q)$ *with minimal* q *where* $X_i \in \mathbf{X}$ *for* $i = 1, \ldots, q$ *is a query to an oracle and* $\mathbf{X}$ *is the set of all possible queries , such that the set of possible diagnoses can be reduced to* $\mathbf{D} = \{\mathcal{D}^*\}$ *where* $\mathcal{D}^* \in \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ *is called the target diagnosis. The order of queries is crucial and affects* q*.*

The function of the mentioned oracle to answer queries is usually realized by the user who created the ontology or by an expert in the domain modeled by the ontology. A set of axioms X_j is called a **query** if it is a subset of common entailments of a set of ontologies $(\mathcal{O} \setminus \mathcal{D}_i)$ where $\mathcal{D}_i \in \mathbf{D}_j^X \subset \mathbf{D}$. In description logics there is a number of different entailment types ET. However, in general not all of these types are used to construct queries as this could produce high numbers of entailments and would make the process of query answering very time-consuming and inefficient. In [9], for example, queries are generated by considering only entailments obtained by classification and realization. A query X_j means asking the oracle $(\mathcal{O}^* \models X_j?)$ and is answered either by *true*, i.e. $\mathcal{O}^* \models X_j$, or by *false*, i.e. $\mathcal{O}^* \not\models X_j$. A query X_j partitions the set of diagnoses $\mathbf{D}$ into $\langle \mathbf{D}_j^X, \mathbf{D}_j^{\neg X}, \mathbf{D}_j^{\emptyset} \rangle$ such that:

- $\forall \mathcal{D}_i \in \mathbf{D}_j^X : \quad (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup (\bigwedge_{t^+ \in TC^+} t^+) \models X_j$,
- $\forall \mathcal{D}_i \in \mathbf{D}_j^{\neg X} : \quad (\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup (\bigwedge_{t^+ \in TC^+} t^+) \models \neg X_j$, and
- $\mathbf{D}_j^{\emptyset} = \mathbf{D} \setminus (\mathbf{D}_j^X \cup \mathbf{D}_j^{\neg X})$.

A query X_j is stored together with its partition as $(X_j, \langle \mathbf{D}_j^X, \mathbf{D}_j^{\neg X}, \mathbf{D}_j^{\emptyset} \rangle)$. Algorithm 1 shows a procedure for computing all queries $\mathbf{X}$ for given $\mathbf{D}$. The function GETENTAILMENTS$(R_E, axioms)$ calls the reasoner R_E to get all entailments X of type $E \in ET$ of *axioms* and returns $\emptyset$ if *axioms* is inconsistent. Applied to our example ontology $\mathcal{O}_{ex}$, this algorithm returns the set of all possible queries shown in Table 1.

If a query X_j is answered positively, then X_j is added to the positive test cases, i.e. $TC^+ \cup \{X_j\}$, and a diagnosis $\mathcal{D}_i \in \mathbf{D}$ is a diagnosis for $\langle \mathcal{O}, \mathcal{B}, TC^+ \cup \{X_j\}, TC^- \rangle$ iff $(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup TC^+ \cup X_j \not\models t^-$ for all $t^- \in TC^-$. If a query X_j is answered negatively, then X_j is added to the negative test cases, i.e. $TC^- \cup \{X_j\}$, and a diagnosis $\mathcal{D}_i \in \mathbf{D}$

Query	$\mathbf{D}_i^X$	$\mathbf{D}_i^{\neg X}$	$\mathbf{D}_i^{\emptyset}$
$X_1 : \{B(w)\}$	$\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6$	$\mathcal{D}_1$	$\emptyset$
$X_2 : \{C(w)\}$	$\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_2$	$\emptyset$
$X_3 : \{D(w)\}$	$\mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$	$\emptyset$
$X_4 : \{E(w)\}$	$\mathcal{D}_5, \mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4$	$\emptyset$
$X_5 : \{F(w)\}$	$\mathcal{D}_6$	$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5$	$\emptyset$

Table 1. Possible queries for diagnoses $\mathcal{D}_i \in \mathbf{D}$

Algorithm 2: Generic Diagnosis Discrimination

Input: diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$, set of diagnoses $\mathbf{D}$, meta information *Info*
Output: target diagnosis $\mathcal{D}^*$

```
repeat
    X ← getBestQuery (D, Info) ;
    if getQueryAnswer (X) == true then D ← D \ D^{¬X};
    else D ← D \ D^X;
until |D| == 1;
return D;
```

is a diagnosis for $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \cup \{X_j\} \rangle$ iff $(\mathcal{O} \setminus \mathcal{D}_i) \cup \mathcal{B} \cup TC^+ \not\models X_j$. So, given that $X_j = \mathit{true}$, the set of eliminated diagnoses is $\mathbf{D}_j^{\neg X}$ and the set of remaining diagnoses is $\mathbf{D}_j^X \cup \mathbf{D}_j^{\emptyset}$. Otherwise, the set of rejected diagnoses is $\mathbf{D}_j^X$ and the set of remaining diagnoses is $\mathbf{D}_j^{\neg X} \cup \mathbf{D}_j^{\emptyset}$.

A generic diagnoses discrimination algorithm is described in algorithm 2. The essential part of this algorithm is the GETBESTQUERY function whose implementation determines its performance decisively. In [9], two different implementations of GETBESTQUERY are studied, namely split-in-half and entropy-based query selection.

The **entropy-based approach** uses meta information about probabilities p_t that the user makes a fault when using a syntactical construct of type $t \in CT$ where CT is the set of construct types available in the used ontology expression language. For example, $\forall$, $\exists$, $\sqsubseteq$, $\neg$, $\sqcup$, $\sqcap$ are some OWL DL construct types. These fault probabilities p_t are assumed to be independent and used to calculate fault probabilities of axioms as

$$p(ax_k) = 1 - \prod_{t \in CT} (1 - p_t)^{n(t)} \tag{1}$$

where $n(t)$ is the number of occurrences of construct type t in ax_k. These probabilities are in turn used to determine fault probabilities of diagnoses $\mathcal{D}_i \in \mathbf{D}$ as

$$p(\mathcal{D}_i) = \prod_{ax_r \in \mathcal{D}_i} p(ax_r) \prod_{ax_s \in \mathcal{O} \setminus \mathcal{D}_i} (1 - p(ax_s)). \tag{2}$$

The strategy is then to select the query which minimizes the expected entropy in the diagnosis probabilities after the query. This means that uncertainty should be minimized and information gain should be maximized. According to [5] this is equivalent to choosing the query X_j which minimizes the following scoring function:

$$sc_{ent}(X_j) = \sum_{a_j \in \{\mathit{true}, \mathit{false}\}} p(X_j = a_j) \log_2 p(X_j = a_j) + p(\mathbf{D}_j^{\emptyset}) + 1 \tag{3}$$

This function is minimized by queries X_j with $p(\mathbf{D}_j^X) = p(\mathbf{D}_j^{\neg X}) = 0.5$. So, entropy-based query selection favors queries whose outcome is most uncertain. After each query X_j, the new information obtained by the feedback of the oracle is incorporated in the meta information by updating the diagnosis probabilities according to the Bayesian formula:

$$p(\mathcal{D}_i|X_j = a_j) = \frac{p(X_j = a_j|\mathcal{D}_i)p(\mathcal{D}_i)}{p(X_j = a_j)} \tag{4}$$

where $a_j \in \{\mathit{true}, \mathit{false}\}$ and

$$p(X_j = \mathit{true}) = \sum_{\mathcal{D}_r \in \mathbf{D}_j^X} p(\mathcal{D}_r) + \frac{1}{2} \sum_{\mathcal{D}_k \in \mathbf{D}_j^\emptyset}$$

and $p(X_j = a_j|\mathcal{D}_k) := 1/2$ for $\mathcal{D}_k \in \mathbf{D}_j^\emptyset$ and in $\{0, 1\}$ for $\mathcal{D}_r \in \mathbf{D}_j^X \cup \mathbf{D}_j^{\neg X}$ according to the explanations on page 4.

The **split-in-half approach**, on the other hand, acts completely independently of any meta information and selects the query X_j which minimizes the following scoring function:

$$sc_{split}(X_j) = \big||\mathbf{D}_j^X| - |\mathbf{D}_j^{\neg X}|\big| + |\mathbf{D}_j^\emptyset|$$

So, the split-in-half strategy prefers queries which eliminate half of the diagnoses, independently of the query outcome.

The result of the evaluation in [9] is that entropy-based query selection reveals better performance than split-in-half if the given meta information is suitable. However, the question is how to choose the fault probabilities profitably in that they favor the unknown target diagnosis. By setting probabilities equally for each user, e.g. according to the results of the study conducted by [6], it cannot be taken for granted that this meta information will be correct for all users. On the contrary, since every user is prone to individual and thus generally different fault patterns, we would rather need empirical data for each individual user in order to set probabilities appropriately. But unfortunately there is neither a log file for each user which could be used as a documentation of common individual faults nor any extensive user-study which would identify user profiles and allow us to set probabilities more individually. Hence, an objective approach to assigning fault probabilities for arbitrary users will generally not bear fruit. As a consequence, one may often need to rely on a vague subjective estimation of the fault probabilities by the user which might not always conform to the de facto fault probabilities. In fact, there are situations where the split-in-half approach outperforms the entropy-based strategy, namely when the fault probabilities disfavor the target diagnosis $\mathcal{D}^*$, i.e. assign low probability to $\mathcal{D}^*$. In this case, entropy-based query selection can be mislead by this poor meta information and might prefer queries that do not favor the target diagnosis.

To illustrate this, let us assume that the user who wants to debug our example ontology $\mathcal{O}_{ex}$ specifies the following fault probabilities: $p_\forall = 0.48$, $p_\exists = 0.11$, $p_\neg = 0.06$, $p_\sqcap = 0.03$, $p_\sqsubseteq = 0.03$. Application of formulas (1) and (2) leads to diagnoses fault probabilities $p(\mathcal{D}_1) = 0.634$, $p(\mathcal{D}_2) = 0.2$, $p(\mathcal{D}_3) = 0.084$, $p(\mathcal{D}_4) = 0.04$, $p(\mathcal{D}_5) = 0.02$, $p(\mathcal{D}_6) = 0.02$. Using entropy-based selection, X_1, i.e. $(\mathcal{O}_{ex}^* \models \{B(w)\}?)$, will be identified as the optimal query since the difference between probabilities of positive

($p(X_1 = \textit{true}) = 0.365$) and negative ($p(X_1 = \textit{false}) = 0.634$) outcome is less than for all other queries $\{X_2, X_3, X_4, X_5\}$ (see Table 1). However, note that this query could eliminate only a single diagnosis, i.e. $\mathcal{D}_1$, if the unfavorable answer is given. Hence, we call X_1 a high-risk query. If, e.g. $\mathcal{D}_5$ is assumed to be the target diagnosis, i.e. $\mathcal{D}^* = \mathcal{D}_5$, then X_1 will be answered positively and all but one diagnoses are still possible. The elimination rate of this query is therefore $1/6$. The probability update given by formula (4) then yields $p(\mathcal{D}_2) = 0.316$, $p(\mathcal{D}_3) = 0.133$, $p(\mathcal{D}_4) = 0.064$, $p(\mathcal{D}_5) = 0.031$, $p(\mathcal{D}_6) = 0.031$. As a next query, X_2 will be preferred, but will be answered unfavorably as well resulting again in the elimination of only one single diagnosis. This procedure can be further executed, i.e. by querying X_3 and X_4, until the target diagnosis $\mathcal{D}_5$ will be identified by getting the answer *false* to query X_5. So, by applying sc_{ent} all five queries are required in order to find the target diagnosis. If queries are selected by sc_{split}, on the contrary, three queries will suffice. At first, query X_3 will be chosen because it eliminates half of all diagnoses in any case. We call such a query a no-risk query. The answer will be *true* and querying X_5 after $X_4 = \textit{true}$ will finally reveal the target diagnosis $\mathcal{D}^* = \mathcal{D}_5$.

This example demonstrates that the no-risk strategy sc_{split} (three queries) is more suitable than sc_{ent} (five queries) for fault probabilities which disfavor the target diagnosis. Let us suppose, on the other hand, that probabilities are assigned reasonably in our example, e.g. that $\mathcal{D}^* = \mathcal{D}_2$. Then it will take the entropy-based strategy only two queries (X_1, X_2) to find $\mathcal{D}^*$ while split-in-half will still require three queries (X_3, X_2, X_1). The complexity of sc_{ent} in terms of required queries varies between $O(1)$ in the best and $O(|\mathbf{D}|)$ in the worst case depending on the appropriateness of the fault probabilities. On the other hand, sc_{split} always requires $O(\log_2 |\mathbf{D}|)$ queries.

We learn from this example that the best choice of discrimination strategy depends on the quality of the meta information. Unfortunately, however, it is impossible to assess the quality of fault probabilities before the debugging session starts since this would require us to know the target diagnosis in advance. Rather, we can exploit the additional information gathered by querying the oracle in order to estimate the quality of probabilities. Let us consider e.g. the first query X_1 selected by sc_{ent} in our example. If this query is answered unfavorably, i.e. by *true*, then many more diagnoses remain than are eliminated. Since the observed outcome was less likely, the partition $\mathbf{D}_1^X$ with lower average diagnosis probability survives. This could be a hint that the probabilities are only weakly justified. The new strategy we present in this work incorporates exactly this information, i.e. the elimination rate achieved by a query, when choosing the successive query by adapting the maximum allowed 'query-risk'. Our method combines the advantages of both the entropy-based approach and the split-in-half approach. On the one hand, it exploits the given meta information *Info* if *Info* is of high quality, but, on the other hand, it quickly loses trust in *Info* and becomes more cautious if some indication is given that *Info* is of poor quality.

3 Risk Optimization Strategy for Query Selection

Our risk optimization algorithm is a dynamic learning procedure which on the one hand learns by reinforcement how to select optimal queries and on the other hand continu-

ally improves the a-priori given meta information based on new knowledge obtained through queries to an oracle. The behavior of our algorithm can be co-determined by the user. Therefore, the algorithm takes into account the user's doubt about the meta information, i.e. the fault probabilities p_t for $t \in CT$, by allowing them to define the initial cautiousness c of the algorithm as well as a cautiousness interval $[\underline{c}, \overline{c}]$ where $c, \underline{c}, \overline{c} \in [0, \lfloor |\mathbf{D}|/2 \rfloor / |\mathbf{D}|]$ and $\underline{c} \leq c \leq \overline{c}$. The interval $[\underline{c}, \overline{c}]$ constitutes the set of all admissible cautiousness values the algorithm may take during the debugging session. High trust in the meta information is reflected by specifying a low minimum required cautiousness $\underline{c}$. If the user is unsure about the rationality of the fault probabilities this can be expressed by setting $\underline{c}$ to a higher value. The value assigned to $\overline{c}$ indicates the maximum admissible cautiousness of the algorithm and can be interpreted as the minimum chance a user wants to preserve of performing better than a no-risk strategy.

This additional cautiousness information guides the behavior of the algorithm when selecting queries. The relationship between cautiousness c and queries is formalized by the following definitions:

Definition 1 (Cautiousness of a Query). *We define the cautiousness $caut(X_i)$ of a query X_i as follows:*

$$caut(X_i) := \frac{min\left\{|\mathbf{D}_i^X|, |\mathbf{D}_i^{\neg X}|\right\}}{|\mathbf{D}|} \in \left[0, \frac{\left\lfloor \frac{|\mathbf{D}|}{2} \right\rfloor}{|\mathbf{D}|}\right]$$

A query X_i is called braver than query X_j iff $caut(X_i) < caut(X_j)$. Otherwise X_i is called more cautious than X_j. A query with highest possible cautiousness is called no-risk query.

Definition 2 (Elimination Rate). *Given a query X_i and the corresponding answer $a_i \in \{true, false\}$, the elimination rate $e(X_i, a_i)$ is defined as follows:*

$$e(X_i, a_i) = \begin{cases} \frac{|\mathbf{D}_i^{\neg X}|}{|\mathbf{D}|} & \text{if } a_i = true \\ \frac{|\mathbf{D}_i^{X}|}{|\mathbf{D}|} & \text{if } a_i = false \end{cases}$$

The answer a_i to a query X_i is called favorable iff it maximizes the elimination rate $e(X_i, a_i)$. Otherwise a_i is called unfavorable.

So, the cautiousness $caut(X_i)$ of a query X_i is exactly the minimal elimination rate, i.e. $caut(X_i) = e(X_i, a_i)$ given that a_i is the unfavorable query result. Intuitively, the user-defined cautiousness c is the minimum percentage of diagnoses in $\mathbf{D}$ which should be eliminated by the successive query. For brave queries the interval between minimum and maximum elimination rate is larger than for cautious queries. For no-risk queries it is minimal.

Definition 3 (High-Risk Query). *Given a query X_i and cautiousness c, then X_i is called a high-risk query iff $caut(X_i) < c$, i.e. the cautiousness of the query is lower than the cautiousness required by the user. Otherwise X_i is called non-high-risk query. By $HR(\mathbf{X}) \subseteq \mathbf{X}$ we denote the set of all high-risk queries. The set of all queries $\mathbf{X}$ can be partitioned in high-risk queries and non-high-risk queries.*

Example 1. Reconsider the example ontology $\mathcal{O}_{ex}$ of Section 2 and let $c = 0.25$. Then query X_2 has cautiousness $caut(X_2) = 2/6 = 0.33 \geq 0.25 = c$ and is thus a non-high-risk query. Query X_1, on the contrary, is a high-risk query because $caut(X_1) = 1/6 = 0.17 < 0.25 = c$. However, if X_1 is not asked as first but for example as third query after $\mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6$ have been eliminated by X_3 and X_2 (compare with the behavior of split-in-half for $\mathcal{O}_{ex}$ in Section 2), the cautiousness $caut(X_1) = 0.5$ which means that X_1 is a non-high-risk query in this case.

Algorithm 3 describes our Risk Optimization Algorithm. Its core is the implementation of the GETBESTQUERY function in Algorithm 2 which is explained in the following (with the denotation of the respective method in brackets). Further details concerning Algorithm 3 are provided in Section 4.

1. (GETMINSCOREQUERY) Determine the best query $X_i \in \mathbf{X}$ according to sc_{ent}. That is:
$$X_i = \arg\min_{X_k \in \mathbf{X}}(sc_{ent}(X_k)).$$
2. (GETQUERYCAUTIOUSNESS) If X_i is a non-high-risk query, i.e. $caut(X_i) \geq c$, select X_i. In this case X_i is the query with maximum information gain among all queries $\mathbf{X}$ and additionally guarantees the required elimination rate specified by c.
3. (GETALTERNATIVEQUERY) Otherwise, select the query $X_j \in \mathbf{X}$ $(X_j \neq X_i)$ which has minimal score sc_{ent} among all least cautious non-high-risk queries L. That means:
$$X_j = \arg\min_{X_k \in L}(sc_{ent}(X_k)).$$
where $L = \{X_r \in \mathbf{X} \setminus HR(\mathbf{X}) \mid \forall X_t \in \mathbf{X} \setminus HR(\mathbf{X}) : \ caut(X_r) \leq caut(X_t)\}$. If there is no such query $X_j \in \mathbf{X}$, then select X_i from step 1.
4. Given the query outcome $X_s = a_s$, update the meta information as per steps 4a and 4b. If step 2 was successful, then $X_s = X_i$, otherwise $X_s = X_j$.
   (a) (UPDATECAUTIOUSNESS) Update the cautiousness c depending on the elimination rate $e(X_s, a_s)$ as follows:
$$c \ \leftarrow \ c + c_{adj} \tag{5}$$
   where c_{adj} denotes the cautiousness adjustment factor which is defined as follows:
$$c_{adj} \ := \ adj \ 2\,(\overline{c} - \underline{c}) \quad \text{where} \quad adj := \frac{\left\lfloor \frac{|\mathbf{D}|}{2} - \epsilon \right\rfloor}{|\mathbf{D}|} - e(X_s, a_s) \tag{6}$$
   and $\epsilon \in (0, \frac{1}{2})$ an arbitrary real number. The factor $2\,(\overline{c} - \underline{c})$ in formula (6) simply regulates the extent of the cautiousness adjustment depending on the interval length $\overline{c} - \underline{c}$. The more crucial factor in the formula is adj which indicates whether c_{adj} is positive or negative, i.e. whether the cautiousness is increased or decreased. Note that $adj < 0$ holds for any favorable query result, independently of the cautiousness $caut(X_s)$ of the asked query X_s. This has the following reasons: For even $|\mathbf{D}|$, the elimination rate resulting from the favorable

query outcome is $\geq \lfloor \frac{|\mathbf{D}|}{2} \rfloor / |\mathbf{D}|$, which is the maximal value the minimal elimination rate of a query can take (see Definition 1). But, also $\lfloor \frac{|\mathbf{D}|}{2} - \epsilon \rfloor / |\mathbf{D}| < \lfloor \frac{|\mathbf{D}|}{2} \rfloor / |\mathbf{D}|$ holds since $|\mathbf{D}|$ is even. Given that $|\mathbf{D}|$ is odd, the elimination rate for any favorable query result is $> \lfloor \frac{|\mathbf{D}|}{2} \rfloor / |\mathbf{D}| = \lfloor \frac{|\mathbf{D}|}{2} - \epsilon \rfloor / |\mathbf{D}|$. If the value c computed in (5) is outside the user-defined cautiousness interval $[\underline{c}, \overline{c}]$, it is set to $\underline{c}$ if $c < \underline{c}$ and to $\overline{c}$ if $c > \overline{c}$. Positive c_{adj} is a penalty telling the algorithm to get more cautious, whereas negative c_{adj} is a bonus resulting in a braver behavior of the algorithm.

(b) (GETPROBABILITIES) Update the diagnosis probabilities $p(\mathcal{D}_r)$ for $\mathcal{D}_r \in \mathbf{D}$ according to formula (4) on page 6. Section 4 gives more details on this method.

To illustrate the behavior of our algorithm, we revisit the example ontology $\mathcal{O}_{ex}$ with six diagnoses discussed in Section 2. Again, we first assume that $\mathcal{D}^* = \mathcal{D}_2$. Further on, let the user be quite unsure whether to trust or not trust in the fault probabilities (see page 6) and thus define cautiousness $c := 0.3$ and the cautiousness interval $[\underline{c}, \overline{c}] := [0, \lfloor \frac{6}{2} \rfloor / 6] = [0, 0.5]$. Then, our algorithm will first test if the query X_1 which has minimal score sc_{ent} is admissible w.r.t. c. Therefore, $caut(X_1) = 1/6 = 0.17$ is calculated and compared with $c = 0.3$. Since $caut(X_1) < c$, X_1 is denied. As a consequence, the algorithm seeks the query with minimal sc_{ent} among all least cautious non-high-risk queries, i.e. those which guarantee elimination of a minimal number $\geq \lceil 0.3 * 6 \rceil = 2$ diagnoses. The only least cautious non-high-risk query is X_2, so it is chosen as first query and answered by *false*. Therefore, the set of possible diagnoses is reduced to $\mathbf{D} = \{\mathcal{D}_1, \mathcal{D}_2\}$. Then, the only query allowing to discriminate between $\mathcal{D}_1$ and $\mathcal{D}_2$, namely X_1, is selected and $\mathcal{D}^* = \mathcal{D}_2$ is identified. For $\mathcal{D}^* = \mathcal{D}_5$, the algorithm acts as follows: The first query is again X_2 which is answered by *true* in this case. Hence, two diagnoses $\mathcal{D}_1$ and $\mathcal{D}_2$ are dismissed yielding an elimination rate $e(X_2, \textit{true}) = 2/6$. Applying formula (6) then yields $c_{adj} = 0$, i.e. c remains unchanged. Thus the next query should eliminate at least $\lceil 0.3 * (6 - 2) \rceil = 2$ diagnoses. The only candidate X_4 (=*true*) is selected, leaving open only one possible last query X_5. So, in both situations $\mathcal{D}^* = \mathcal{D}_5$ (three queries) and $\mathcal{D}^* = \mathcal{D}_2$ (two queries) our method performed as well as the better of sc_{ent} and sc_{split}, respectively.

4 Implementation Details

Our Risk Optimization Algorithm (Algorithm 3) takes the following inputs: (1) A diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$, (2) meta information *Info*=(P, C), comprising on the one hand fault probabilities $P = \{p_t | t \in CT\}$ where CT is the set of syntactical construct types available in the used ontology expression language, and on the other hand and user-defined cautiousness parameters $C = (c, \underline{c}, \overline{c})$, (3) the number n of considered leading diagnoses per iteration, and (4) a stopping criterion *stop_cond*. Input 3 is needed because, in general, it is impossible to consider all minimal diagnoses for a given diagnosis problem instance at once. A simple way of alleviating this problem is to consider only minimal diagnoses since the set of all diagnoses is sufficiently described by the set of all minimal diagnoses. This holds because any superset of a diagnosis is also a diagnosis due to the fact that eliminating an axiom from a consistent

Algorithm 3: Risk Optimization Algorithm

Input: diagnosis problem instance $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$, meta information *Info*=(P, C) where $P = \{p_t | t \in CT\}$ and $C = (c, \underline{c}, \overline{c})$, size of leading diagnoses window n, stop condition *stop_cond*
Output: a diagnosis $\mathcal{D}$

$TC^+ \leftarrow \emptyset$; $TC^- \leftarrow \emptyset$; $\mathbf{D} \leftarrow \emptyset$;
repeat
  $\mathbf{D} \leftarrow$ `getDiagnoses`$(\mathcal{O}, \mathcal{B}, TC^+, TC^-, n, \mathbf{D})$;
  $P \leftarrow$ `getProbabilities`$(P, \mathbf{D}, TC^+, TC^-)$;
  $\mathbf{X} \leftarrow$ `generateQueries`$(\mathcal{O}, \mathcal{B}, TC^+, \mathbf{D})$;
  $X_i \leftarrow$ `getMinScoreQuery`$(P, \mathbf{X}, sc_{ent})$;
  if `getQueryCautiousness`$(X_i, \mathbf{D}) < c$ **then** $X_i \leftarrow$ `getAlternativeQuery`$(c, \mathbf{X}, P, \mathbf{D})$;
  if `getAnswer`$(X_i) ==$ *true* **then** $TC^+ \leftarrow TC^+ \cup \{X_i\}$;
  else $TC^- \leftarrow TC^- \cup \{X_i\}$;
  $c \leftarrow$ `updateCautiousness`$(\mathbf{D}, TC^+, TC^-, X_i, c, \underline{c}, \overline{c})$;
until (*stop_cond* $\vee$ `getScore`$(X_i, sc_{ent}) == 1$);
return `mostProbableDiag`$(\mathbf{D}, P)$;

ontology cannot make the ontology inconsistent. However, this does not solve the problem as there might also be a huge number of minimal diagnoses. The problem with a large number of diagnoses is caused by the time complexity needed to find diagnoses. In order to explain this problem in more detail, we first provide a short review of the diagnosis computation method described in [1] which we employ to calculate diagnoses. This method exploits the notion of a conflict set which is defined as follows:

Definition 4 (Conflict Set). *Given a diagnosis problem $\langle \mathcal{O}, \mathcal{B}, TC^+, TC^- \rangle$, a conflict set $CS \subseteq O$ is a set of axioms s.t. there is a $t^- \in TC^-$ for which $CS \cup \mathcal{B} \cup TC^+ \cup t^-$ is inconsistent. A conflict set CS is called minimal iff there is no $CS' \subset CS$ such that CS' is a conflict set.*

Simply put, a conflict set CS is an inconsistent part of the ontology, i.e. CS does not meet requirements 1 or 3 in Problem Definition 1. If a diagnosis problem instance does not include any conflict set, then this problem instance is either solved, i.e. meets requirements 1, 2 and 3 in Problem Definition 1, or can be easily solved by adding the desired positive test cases TC^+ if requirement 2 is not met. So, the idea is to resolve every conflict set in order to solve a given diagnosis problem instance. This is achieved by deleting or changing at least one axiom of a *minimal* conflict set. The following proposition [7] summarizes these thoughts:

Proposition 1. *$\mathcal{D} \subseteq \mathcal{O}$ is a (minimal) diagnosis iff $\mathcal{D}$ is a (minimal) hitting set of all minimal conflict sets.*

In [1], the QUICKXPLAIN algorithm (in short QX) introduced by [2] is used to compute minimal conflict sets. $QX(B, A)$ takes as inputs a set of trusted background knowledge axioms B and a set of axioms A and returns a minimal conflict set $CS \subseteq A$ w.r.t. B. If $A \cup B$ is consistent, QX outputs *consistent*. If B is inconsistent, the output is $\emptyset$. In order to compute minimal diagnoses, a tree called HS-Tree is built with minimal conflict sets as nodes. The path from the root node nd^0 to a node nd_i^k on level k is denoted by $HS(nd_i^k)$. The root of this tree is a minimal conflict set obtained by calling $QX(\mathcal{B} \cup TC^+ \cup t^-, \mathcal{O})$ for $t^- \in TC^-$ until it first outputs $CS(nd^0) \neq$ *consistent*. For each axiom $ax_i \in CS(nd_j^{k-1})$ there is a path $HS(nd_i^k) = HS(nd_j^{k-1}) \cup \{ax_i\}$ to

a node nd_i^k. In order to compute a minimal conflict set for node nd_i^k, $QX(\mathcal{B} \cup TC^+ \cup t^-, \mathcal{O} \setminus HS(nd_i^k))$ is run for $t^- \in TC^-$ until an output $CS(nd_i^k) \neq$ *consistent* is returned. If all outputs are *consistent*, i.e. $\mathcal{O} \setminus HS(nd_i^k) \cup \mathcal{B} \cup TC^+ \cup t^-$ is consistent for all $t^- \in TC^-$, then a minimal hitting set $\mathcal{D} = HS(nd_i^k)$ of all minimal conflict sets, i.e. a minimal diagnosis, is found and node nd_i^k is not further expanded.

The calculation of all minimal diagnoses at once would thus involve the complete construction of the HS-Tree. However, in the worst case this requires an exponential (in n_{CS}) number of calls to $QX(B, A)$ which itself requires $O(\log_2 |A|)$ calls to an underlying reasoner where n_{CS} denotes the total number of conflict sets. Also, generating all queries (w.r.t. certain entailment types) for a number k of diagnoses quickly becomes impracticable as k grows, since all non-trivial $2^k - 2$ partitions are explored (see Algorithm 1). Therefore it is generally not feasible to examine all minimal diagnoses at the same time.

In our approach, we tackle this issue by introducing a leading diagnoses window of fixed size n, i.e. $|\mathbf{D}| = n$. That is, when the debugging session starts, the first n minimal diagnoses are computed as explained before. However, instead of calculating the n minimal diagnoses by ascending cardinality as in [9], our method implements a uniform-cost search on the HS-Tree to find the n most probable minimal diagnoses. This uniform-cost search is guided by the meta information, i.e. the fault probabilities $P = \{p_t | t \in CT\}$, provided as an input to our algorithm. From these fault probabilities, the probabilities $p(ax_j)$ of axioms $ax_j \in \mathcal{O}$ being erroneous can be determined according to formula (1). These probabilities are then used to decide which node to expand next in the search. Namely, always the node $nd_i \in V$ with path $HS(nd_i)$ from the root node is expanded where $p(HS(nd_i)) = \prod_{ax_j \in HS(nd_i)} p(ax_j)$ is maximal for nd_i among all nodes $nd_k \in V$ where V is the set of all generated nodes in the HS-Tree. In this manner, most probable diagnoses are generated first by the GETDIAGNOSES function. If $g \leq n$ leading diagnoses are eliminated by a query, then, in the next iteration, the function supplies the next most probable $n - g$ diagnoses, and so on. Visually spoken, one could say that the leading diagnoses window moves on for $e(X_i, a_i)\,|\mathbf{D}|$ steps after each query X_i to include the next most probable $e(X_i, a_i)\,|\mathbf{D}|$ diagnoses.

The GETPROBABILITIES function in each iteration calculates the diagnosis probabilities for the current set of leading diagnoses $\mathbf{D}$ taking into account all information gathered by querying the oracle so far. To that end it calculates the adjusted diagnosis probabilities $p_{adj}(\mathcal{D}_i)$ for $\mathcal{D}_i \in \mathbf{D}$ as $p_{adj}(\mathcal{D}_i) = (1/2)^z\, p(\mathcal{D}_i)$ where z is the number of precedent queries X_k where $\mathcal{D}_i \in \mathbf{D}_k^{\emptyset}$ and $p(\mathcal{D}_i)$ is the probability of $\mathcal{D}_i$ calculated directly from the preliminarily given fault probabilities P by formulas (1) and (2). Afterwards the probabilities $p_{adj}(\mathcal{D}_i)$ are normalized. Note that z can be computed from TC^+ and TC^- which comprise all query answers. This way of updating probabilities is exactly in compliance with the Bayesian theorem given by formula (4). The rest of the methods used in Algorithm 3 is explained in Section 3.

5 Evaluation

We tested our Risk Optimization Algorithm (R) against the entropy-based approach (E) and the split-in-half approach (S) which were explained in Section 2. As adaptiveness is one of the central achievements of our approach, the main goal of this evaluation is

to show its robustness in a variety of possible application scenarios. A major obstacle, however, is that only a small number of inconsistent/incoherent ontologies are available and they capture only a few situations, like ontology learning or tutorial cases. So, we decided to simulate diagnosis sessions covering a broader variety of situations using a model M which takes as inputs a set of total diagnoses, fault probabilities, and the parameters explained throughout the rest of this section. M is *realistic* because the values of parameters used in M (e.g. for query generation and diagnosis elimination) were determined by taking averages (*) after analyzing the structure and dependencies between diagnoses as well as between diagnoses and queries using real-world inconsistent/incoherent ontologies as in [9]. M is *fair* since each approach $A \in \{E, S, R\}$ was tested under exactly the same conditions (i.e. diagnoses, queries, probabilities, etc.) by means of random seeds. And M *enables exhaustive automated tests*.

We randomly instantiated M in each test iteration featuring different types of fault probability distributions $PT \in \{extreme, moderate, uniform\}$ as well as different cases in terms of quality of probabilities $QU \in \{good, average, bad\}$. Each approach A was tested for the nine different settings in $PT \times QU$. To construct different types of fault probability distributions PT, it is sufficient to change the magnitude of the random ratio $r_{ij} = p(\mathcal{D}_i)/p(\mathcal{D}_j)$ between *diagnosis probabilities* $p(\mathcal{D}_i) > p(\mathcal{D}_j)$ under the assumption that $\mathcal{D}_i$ is a neighbor to $\mathcal{D}_j$ when diagnoses $\mathcal{D}_i$ are ordered descending by $p(.)$. This is because (i) our approach uses the fault probabilities p_t only at the very beginning to calculate the diagnoses probabilities $p(\mathcal{D}_i)$ and subsequently only works with probabilities $p(\mathcal{D}_i)$, and because (ii) our implementation of the diagnosis generation supplies diagnoses in order w.r.t. their probability (see Section 5). So, *extreme*, *moderate* and *uniform* refer to a 'high', 'medium' and equal-to-one r_{ij}, respectively. Concretely, an *extreme* distribution was instantiated by generating uniformly distributed random values $r_i \in [0,1)$, one for each diagnosis $\mathcal{D}_i$, and weighting each r_i by $w_i = (1/1.5)^i$. That is, $p(\mathcal{D}_i) = r_i\, w_i$. As a second step, the probabilities $p(\mathcal{D}_i)$ were normalized and diagnoses ordered descending by $p(.)$. A *moderate* distribution was generated analogously, but with different weights $w_i = (1/1.1)^i$. To obtain a *uniform* distribution we simply assigned equal probabilities to all diagnoses. The quality QU of fault probabilities is measured in terms of the position of the target diagnosis $\mathcal{D}^*$ in the list of all diagnoses ordered by descending probability. By *good*, *average* and *bad* quality we mean that $\mathcal{D}^*$ is located randomly within the first, fourth and ninth tenth of this ordered list.

Our test setting, implemented in Java[2], was the following. For each A and each precondition in $PT \times QU$ we performed 35 iterations in each of which we instantiated M with 200 diagnoses in total. As explained before, we then assigned probabilities to these diagnoses according to PT and we marked a diagnosis as the target $\mathcal{D}^*$ depending on QU. Further on, we specified the number of leading diagnoses to 9 which constitutes a trade-off between computational complexity (e.g. query generation) and amount of information taken into account for query selection. The stop condition *stop_cond* was reasonably defined as follows: If the currently most probable diagnosis $\mathcal{D}_1$ has probability at least three times as high as the second most probable diagnosis $\mathcal{D}_2$, i.e. $p(\mathcal{D}_1) \geq 3\, p(\mathcal{D}_2)$, the user is asked to check if $\mathcal{D}_1$ corresponds to the wanted target diagnosis $\mathcal{D}^*$. If so, $\mathcal{D}^*$ is found and the debugging session ends. Otherwise, the user

[2] See http://rmbd.googlecode.com

continues the debugging session. In our implementation this 'user check' was simply done by testing if $\mathcal{D}_1 == \mathcal{D}^*$. Queries were then generated in conformance with Algorithm 1, where $\mathbf{D}$ is the current set of leading diagnoses. In order to simulate the non-existence of certain queries according to (*), we ran through all subsets $\mathbf{D}_i^X \in \mathbf{D}$ and selected a $\mathbf{D}_i^X$ with probability 0.4 and for a selected $\mathbf{D}_i^X$ we ran through all partitions $\langle \mathbf{D}_i^{\neg X}, \mathbf{D}_i^{\emptyset} \rangle$ of $\mathbf{D} \setminus \mathbf{D}_i^X$ and selected $\mathbf{D}_i^{\neg X}, \mathbf{D}_i^{\emptyset}$ with probability 0.1 to constitute a query $\langle \mathbf{D}_i^X, \mathbf{D}_i^{\neg X}, \mathbf{D}_i^{\emptyset} \rangle$ which was then added to $\mathbf{X}$. The answer to a query X_i is well-defined, if $\mathcal{D}^* \in \mathbf{D}_i^X$ (*true*) or $\mathcal{D}^* \in \mathbf{D}_i^{\neg X}$ (*false*). Otherwise, we simulated the feedback of the oracle in that we rationally generated another diagnosis probability distribution p_{real} which can be seen as the (unknown) 'real' fault distribution which applies for a user. For those queries X_i where $\mathcal{D}^* \notin (\mathbf{D}_i^X \cup \mathbf{D}_i^{\neg X})$, the oracle was implemented to give answers according to p_{real}. The type of the distribution p_{real} was determined by QU. Hence, for the $good$ case, p_{real} was generated in a way that it correlates positively with p, i.e. $p_{\text{real}}(\mathcal{D}_i) = r_i\, p(\mathcal{D}_i)$ where $r_i \in [0, 1)$ is a uniformly distributed random number. For the $average$ case, p_{real} was generated independently of p, i.e. $p_{\text{real}}(\mathcal{D}_i) = r_i$. For the bad case, p_{real} was generated s.t. it correlates negatively with p, i.e. $p_{\text{real}}(\mathcal{D}_i) = r_i / p(\mathcal{D}_i)$. Then these probabilities were normalized.

The relation between axioms in terms of common entailments was realized as follows: For each query $X \in \mathbf{X}$, we specified according to (*) that 95% of diagnoses currently not in the set of leading diagnoses $\mathbf{D}$ make no prediction about the query outcome, i.e. are in no case eliminated by this query. All other $\mathcal{D}_i \notin \mathbf{D}$ were allocated randomly to predict either *true* or *false*.

After each iteration we measured the number of queries $Q\#$ needed to find the target diagnosis $\mathcal{D}^*$, the average elimination rate $ER(\%)$ observed for these $Q\#$ queries, and the number of queries $LD\#$ still needed to find $\mathcal{D}^*$ after it has become a leading diagnosis, i.e. $\mathcal{D}^* \in \mathbf{D}$. Figure 1 illustrates the overall test results. In the bar charts, the figure shows $ER(\%)$, $Q\#$ and $LD\#$ averaged over all 35 iterations for each approach $A \in \{E, S, R\}$ and each precondition in $PT \times QU$. The box plot, on the other hand, indicates the variance of the observed number of queries $Q\#$ during all 35 iterations by providing the minimal and maximal observed values, the first and third quartile and the median. The area between 0.25 and 0.75 quantiles is marked as a box which includes a horizontal line giving the position of the median. We tested our method R with many different settings for cautiousness parameters. Performance was similar for all these settings thanks to the learning algorithm. Evaluation results are given for $(c, \underline{c}, \overline{c}) = (0.4, 0.3, 0.5)$ which yielded the most stable results. In Figure 1 the superior average performance of R in comparison to both other approaches S and E is clearly manifested. It can be observed that R needs fewest queries in each situation. Responsible for this is the high $ER(\%)$ and the low $LD\#$ achieved. Indeed, our method maximizes $ER(\%)$ while minimizing $LD\#$ in all cases compared to S and E. As a consequence of the high $ER(\%)$, the leading diagnoses window in our method moves on quickly to consider less probable diagnoses for discrimination. In this way, the target diagnosis gets 'covered' earlier by the leading diagnoses window and is thus identified with fewer queries. This high $ER(\%)$ can be explained by the property of our method to constantly monitor the elimination rate of selected queries and the ability to react quickly to poor performance by switching to a more cautious strategy. Account-

able for the minimization of $LD\#$ is the learning of probabilities adopted from the entropy-based approach coupled with the high $ER(\%)$ achieved by our method. The improvement of R w.r.t. $Q\#$ is most drastically visible in the box plots which show that the interval of the most frequent 50% of observed $Q\#$ values (denoted by the box) is always located lower or as low as for the better approach of S and E. In case the lower bound of the interval is located as low as for another method, the length of the interval, i.e. the variance, is lower for our method R.

6 Conclusion

In this paper we presented a risk optimization approach to sequential debugging of ontologies. The proposed method exploits meta information in terms of fault probabilities and user-defined willingness to take risk. A substantial cost reduction compared to existing methods was shown for any quality of the given meta information. The proposed method is of particular significance for domains where only vague meta information is available, such as ontology development.

References

1. Friedrich, G., Shchekotykhin, K.: A General Diagnosis Method for Ontologies. In: Gil, Y., Motta, E., Benjamins, R., Musen, M. (eds.) Proceedings of the 4 th International Semantic Web Conference (ISWC-05). pp. 232–246. Springer (2005)
2. Junker, U.: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In: Association for the Advancement of Artificial Intelligence. pp. 167–172. San Jose, CA, USA (2004)
3. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all Justifications of OWL DL Entailments. In: Proc. of ISWC/ASWC2007, Busan, South Korea. LNCS, vol. 4825, pp. 267–280. Springer Verlag, Berlin, Heidelberg (Nov 2007), http://iswc2007.semanticweb.org/papers/267.pdf
4. Kalyanpur, A., Parsia, B., Sirin, E., Cuenca-Grau, B.: Repairing Unsatisfiable Concepts in OWL Ontologies. In: 3rd European Semantic Web Conference, ESWC 2006. Lecture Notes in Computer Science, vol. 4011, pp. 170–184. Springer, Berlin, Heidelberg (2006), http://www.springerlink.com/index/10.1007/11762256
5. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artificial Intelligence 32(1), 97–130 (Apr 1987), http://linkinghub.elsevier.com/retrieve/pii/0004370287900634
6. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) Engineering Knowledge in the Age of the SemanticWeb 14th International Conference, EKAW 2004. pp. 63–81. Springer, Whittenbury Hall, UK (2004), http://www.springerlink.com/index/PNRG2T506C2JV3MD.pdf
7. Reiter, R.: A Theory of Diagnosis from First Principles. Artificial Intelligence 23, 57–95 (1987)
8. Schlobach, S., Huang, Z., Cornet, R., Harmelen, F.: Debugging Incoherent Terminologies. Journal of Automated Reasoning 39(3), 317–349 (2007), http://www.springerlink.com/index/10.1007/s10817-007-9076-z
9. Shchekotykhin, K., Friedrich, G.: Query strategy for sequential ontology debugging. In: Patel-Schneider, P.F., Yue, P., Hitzler, P., Mika, P., Lei, Z., Pan, J., Horrocks, I., Glimm, B. (eds.) Proceedings of International Semantic Web Conference (ISWC 2010). Shanghai, China (2010)

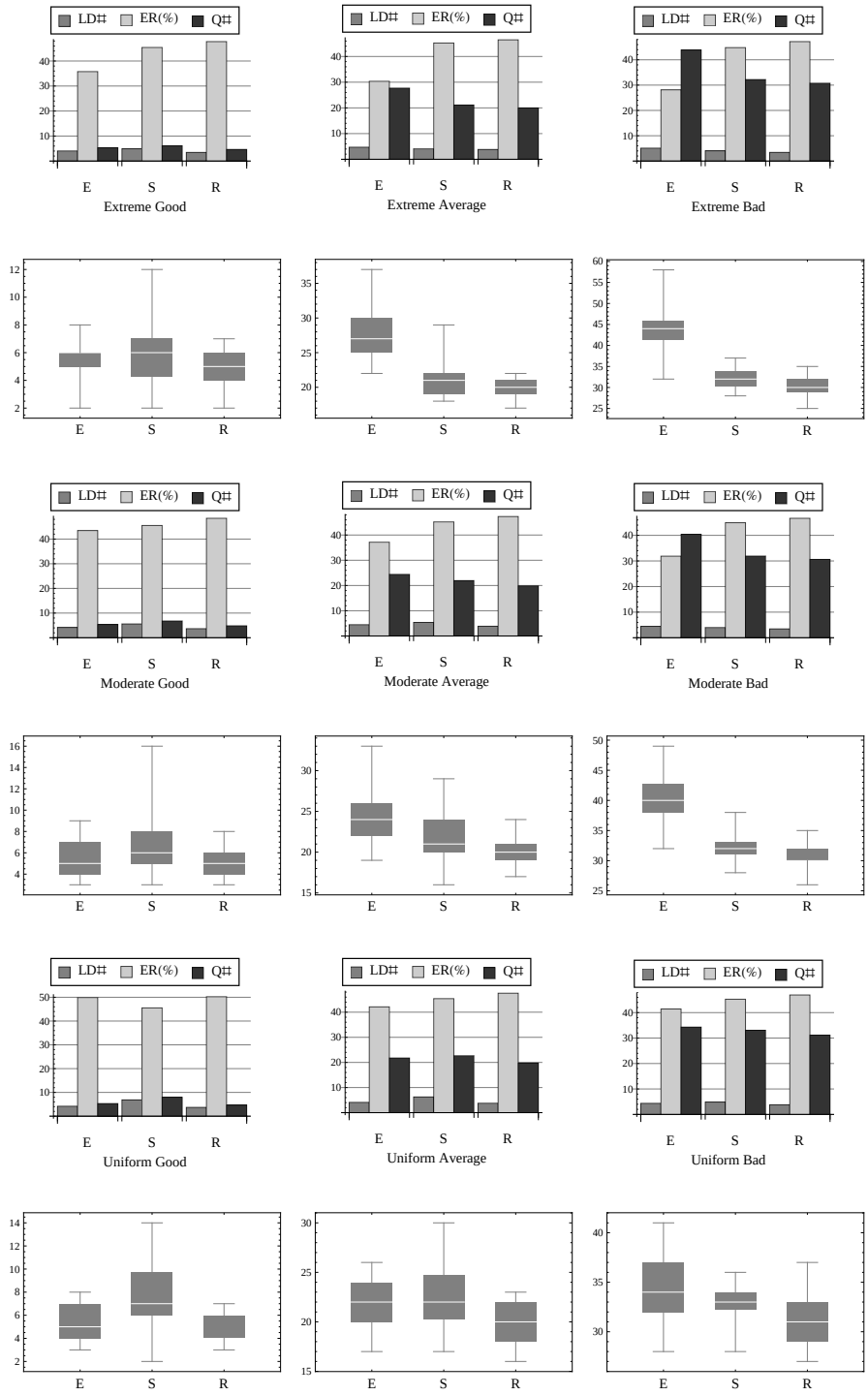

Fig. 1. For each combination of probability distribution type in {*Extreme, Moderate, Uniform*} and quality in {*Good, Average, Bad*} one bar chart and one box plot is shown. Every plot depicts values for approaches S (split-in-half), E (entropy) and R (new method). **Bar charts** show average required queries ($Q\#$), elimination rate ($ER(\%)$) and queries needed after $\mathcal{D}^* \in \mathbf{D}$ ($LD\#$). **Box plots** show the variance of $Q\#$.