On Relaxing Failing Queries over RDF Databases

Paper ID: #5175

Abstract

In this paper, we present a novel approach, CADER, for relaxing failed queries over RDF databases. We show how the number of required database queries for determining all the possible relaxations can be limited to the search of failing subqueries of the user query. Then, we point out how the hitting set problem can be applied for determining all the possible relaxed queries in an efficient way without querying the RDF database. Finally, the scalability and efficiency of our system are shown through extensive experiments on the well-known RDF benchmarks with a variety of queries of different shapes.

1 Introduction

The proliferation of online databases leads to an unprecedented wealth of information that is accessible via the Web. RDF (Resource Description Framework) is a declarative data model for the Semantic Web. With the large scale of RDF data, specialized databases, called *RDF databases* (or triple-stores), have been developed to manage large amounts of RDF data. Many knowledge bases are now defined in the RDF format, e.g., DBpedia [Bizer *et al.*, 2009], Knowledge Vault [Dong *et al.*, 2014], Bio2RDF [Belleau *et al.*, 2008], Yago2 [Hoffart *et al.*, 2013], UniProt [Boutet *et al.*, 2007], etc. and they shape a very large set of graphs having millions of triples interlinked each other. However, the nature of RDF data structure generally prevents users from correctly formulate queries that return a result for their desired requests. This is why user RDF queries often return an empty result.

In this context, failing queries is a key problem where users ignore whether their request are too selective, well formulated or if the expected results are simply not present in the base. A promising method for efficiently querying RDF databases consists of relaxing failing RDF queries. *Query relaxation*, i.e., the removal of parts from the original user query, is beside similarity-based retrieval, one of the commonly used techniques to deal with failing queries problem. The main goal is to apply some transformations in basic queries produced by the user in order to acquire approximate answers for his/her initial need. This is especially important in applications such as information retrieval, where getting an alternative answer may be better than not getting an answer at all. Moreover, in recommendation systems query relaxation can help the user to autonomously decide what is the best compromise when not all his wants and needs can be satisfied.

It is worth to mention that manually relaxing failed queries, which do not match any tuple in an RDF database, is a frustrating, tedious, time-consuming process. Automated query relaxation algorithms

Due to this complexity, the following requirements must be considered when dealing with the query relaxation problem:

- **Completeness (R1):** the solution must provide all the possible relaxed queries;
- **Explicability (R2):** it is useful for the user to provide him with all the causes of failure in order to determine possible explanations for potentially wrong queries;
- **Scalability (R3):** the solution must provide an answer even for complex queries with a high number of triple patterns over large databases;
- **Performance (R4):** the solution must have a fast response time to enable users to retrieve the desired data.

To the best of our knowledge, only one work exists in the literature that proposed two approaches called Lattice-Based Approach (LBA) and Matrix-Based Approach (MBA) for the purpose of MFSs (Minimal Failing Subqueries) and XSSs (maXimal Succeeding Subqueries) computation in the RDF context [Fokou *et al.*, 2017]. Even though these approaches were shown to be successful, they only deal with limited requirements discussed above (i.e., R1 and R2 under some conditions). Indeed, they are complete (R1) and explicable (R2) to some extent as no answer is given for queries beyond 10 triple patterns in a reasonable response time ($< 100s$). In practice, the results differ in their semantics according to the nature and the structure of the query (star, chain or composite). Besides, the relaxation calculus (i.e., finding all XSSs) is extremely linked to the causes of failures' one (finding all MFSes) such that it is not possible to provide the answers independently. In addition, LBA and MBA are less interested in delivering answers in a bounded runtime (R3) and they have a high computational cost, which comes from evaluating a large number of queries against the entire RDF database to compute the possible relaxations (R4).

The contributions we bring to the problem of RDF query relaxation can be outlined as follows:

- An algorithm to find all minimal failing subsets of the initial failing user query;
- A novel method to model the query relaxation problem as a hitting set problem;
- An efficient algorithm for computing all the possible relaxations based on the strong relationship between minimal failing subsets and minimal hitting set problem;
- Extensive experiments on different benchmark RDF datasets, DBpedia and LUBM, with various sets of queries to show the scalability and efficiency of our approach.

These contributions are the basis of our approach named CADER (**C**omplete **A**pproach for R**D**F Qu**E**ry **R**elaxation) that fulfills the aforementioned requirements. Indeed, it is complete, explicable and scalable as it is able to deliver an answer in a linear response time for large RDF datasets (1 billion of triple patterns). Furthermore, we compared the efficiency of our approach against the previously fastest systems LBA and MBA [Fokou *et al.*, 2017], where we demonstrate that CADER is consistently faster and able to speed up the computation to several orders of magnitude. The proofs are not included in this paper due to space constraints.

2 Preliminaries

In this section, we provide the basic terminology that will be used in the paper.

RDF. Given a set U of URI references, a set L of literals (constants), and a set B of blank nodes (unknown URIs or literals) such that U, B and L are pairwise disjoint, an RDF database consists of a set of triples $\langle s, p, o\rangle$ where we refer to $s \in U \cup B$ as *subject*, $p \in U$ as *predicate*, and $o \in U \cup B \cup L$ as *object*. Let V be an infinite set of variables, disjoint from U, B, and L. A *triple pattern* t is an element of $(U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$. $var(t)$ denotes the set of variables occurring in t.

Queries. We consider conjunctive RDF queries (e.g., conjunctive SPARQL queries), a central class of queries in database research and widely considered in research but also in real-world applications [Lanti *et al.*, 2015; Stefanoni *et al.*, 2018]. A conjunctive RDF query (or simply *RDF query*) Q is defined as a *conjunction* of triple patterns, i.e., $Q = t_1 \wedge \ldots \wedge t_n$. Given a query $Q = t_1 \wedge \ldots \wedge t_n$, a query $Q' = t_i \wedge \ldots \wedge t_j$ is a *subquery* of Q iff $\{t_i, \ldots, t_j\} \subseteq \{t_1, \ldots, t_n\}$. In addition, Q' is a *proper superquery* of the query Q iff $\{t_1, \ldots, t_n\} \subset \{t_i, \ldots, t_j\}$.

To define the semantics of query evaluation, we need to recall some further notation. A *mapping* μ is a partial function $\mu : V \rightarrow U \cup B \cup L$. The domain of μ, denoted by $dom(\mu)$, is the subset of V where μ is defined. Given a mapping μ and a triple pattern t s.t. $var(t) \subseteq dom(\mu)$, we have that $\mu(t)$ is the result of replacing every variable $x \in var(t)$ by $\mu(t)$. A mapping μ_1 is said to be *compatible* with a mapping μ_2, written as $\mu_1 \sim \mu_2$, if for all $x \in dom(\mu_1) \cap dom(\mu_2)$, $\mu_1(x) = \mu_2(x)$. Let Ω_1 and Ω_2 be sets of mappings. Then, the *join* of Ω_1 and Ω_2 is defined as: $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \; and \; \mu_1 \sim \mu_2\}$. Finally, given an RDF database $\mathcal{D}$ and an RDF query Q, the evaluation of Q over $\mathcal{D}$, denoted by $[[Q]]_{\mathcal{D}}$, is recursively defined as follows:

- if Q is a triple pattern, then $[[Q]]_{\mathcal{D}} = \{\mu \mid dom(\mu) = var(t) \; and \; \mu(t) \in \mathcal{D}\}$,
- if Q is $t_1 \wedge t_2$, then $[[Q]]_{\mathcal{D}} = [[t_1]]_{\mathcal{D}} \bowtie [[t_2]]_{\mathcal{D}}$.

A common RDF query problem for users is to face with empty answers, i.e., $[[Q]]_{\mathcal{D}} = \emptyset$. Usually, only some parts of Q are responsible of its failure. Finding such subqueries provides the user with an explanation of the empty result returned and a guide to relax the original query. More formally, such notion of causes of failure can be characterized as follows. Let us first denote by $\Sigma = \{t_1, \ldots, t_n\}$ the set of triple patterns contained in the query $Q = t_1 \wedge \ldots \wedge t_n$. We refer to the conjunction of the set of triple patterns $\{t_1, \ldots, t_n\}$ of Σ as $\bigwedge \Sigma$. Obviously, $\bigwedge \Sigma = Q$.

Definition 1 *Let $\mathcal{D}$ be an RDF database. A Minimal Failing Subset Γ of a query Q, denoted by MFS, is a set of triple patterns s.t. (i) $\Gamma \subseteq \Sigma$, (ii) $[[\bigwedge \Gamma]]_{\mathcal{D}} = \emptyset$, and (iii) $\forall \Gamma' \subset \Gamma : [[\bigwedge \Gamma']]_{\mathcal{D}} \neq \emptyset$.*

That is, an MFS of Q is a subset of triple patterns that is *failing* and *minimal* in the sense that removing any one of its elements makes the remaining set of triples *succeed*. Hence, an MFS can be seen as an irreducible cause of the failing of the original query.

A dual concept of MFS is the notion of *MaXimal Succeeding Subset* of an RDF query, which we define as follows:

Definition 2 *Let $\mathcal{D}$ be an RDF database. A maXimal Succeeding Subset Γ of a query Q, denoted by XSS, is a set of triple patterns s.t. (i) $\Gamma \subseteq \Sigma$, (ii) $[[\bigwedge \Gamma]]_{\mathcal{D}} \neq \emptyset$, and (iii) $\forall \Gamma'$ with $\Gamma \subset \Gamma' \subseteq \Sigma : [[\bigwedge \Gamma']]_{\mathcal{D}} = \emptyset$.*

Given a failing query Q, an XSS Γ for Q is a non-failing subset of Q (i.e. $\bigwedge \Gamma$ is a succeeding subquery) and there exists no other query Q' which is also a non-failing subquery of Q such that $\bigwedge \Gamma$ is a subquery of Q'. Obviously, given a failing RDF query Q and Γ an XSS for Q, then $\bigwedge \Gamma$ is a relaxed query of Q. For convenience, we use XSSes and MFSes to denote the set of all maximal succeeding and minimal failing subsets of Q, respectively. Note that in the worst case the number of XSSes and MFSes is exponential in the number of triple patterns in Q.

Example 1 *A user wants to find ... After issuing an RDF query over LUBM [], the user obtains an empty answer. Let us consider the database inspired by the LUBM Benchmark. Let us consider the following query $Q = t_1 \wedge t_2 \wedge t_3 \wedge t_4 \wedge t_5 \wedge t_6$:*

$$
\begin{array}{ll}
SELECT * \; WHERE \{ & \\
?Y10 \;\; ?Y11 \;\; . & (t_1) \\
?Y11 \;\; ?Y12 \;\; . & (t_2) \\
?Y12 \;\; ?Y13 \;\; . & (t_3) \\
?Y13 \;\; ?Y14 \;\; . & (t_4) \\
?Y14 \;\; ?Y15 \;\; . & (t_5) \\
?Y15 \;\; "false" \} & (t_6)
\end{array}
$$

This query will fail over LUBM since... Then, we have MFSes(Q) = $\{\{t_1\}, \{t_3, t_6\}, \{t_2, t_3, t_4\}, \{t_2, t_3, t_5\}\}$. Hence, XSSes(Q) = $\{\{t_2, t_3\}, \{t_3, t_4, t_5\}, \{t_2, t_4, t_5, t_6\}\}$.

3 An Efficient Complete Approach for RDF Query Relaxation

In this section, we introduce a new efficient and complete technique, coined CADER (**C**omplete **A**pproach for R**DF** Qu**E**ry **R**elaxation), for determining possible relaxations over RDF databases in an efficient way to overcome the limitations of existing approaches [Fokou *et al.*, 2017]. Our method is mainly based on the strong duality between MFSes and XSSes. This duality is shown to be the cornerstone of the efficiency of our approach. CADER identifies the root causes of failure and computes all relaxations when the user query fails. In other words, we propose to evaluate the subqueries of the user query individually in a preprocessing step and base the subsequent computation of relaxations on these intermediate results. Our main idea is based on the fact that, in practice, it is more efficient to find minimal failing subsets of triple patterns than maximal succeeding ones. Further, we show how we can efficiently determine all the possible relaxations from minimal failing subsets directly without querying the RDF database. Besides, extracting minimal failing subsets is helpful in many applications, because it emphasize what is wrong with a failing query, i.e., which parts of the query responsible for the missing possible answers.

In a nutshell, our method can be summarized in the following steps: First, we calculate all minimal failing subsets of the user query before computing the entire set of relaxed queries. After that, we compute the hitting sets of these MFSes. These hitting sets are in fact the complement of all the possible relaxations of the failing user query. Accordingly, the complete set of relaxed queries is computed from the set of hitting sets in a direct way by taking the complement of each hitting set.

3.1 Step 1: Finding all MFSes

The basic idea of this step is to search for all the triple patterns that raise a problem following a logical principle. A failing query could have multiple reasons for its infeasibility. In this case, the query would contain multiple MFSes, and fixing any single MFS may not make it successful. As long as any MFS is presented in the query, it will remain infeasible.

Our approach keeps on searching for reasons of failure by browsing different subqueries of the original failing query. More precisely, our approach finds an MFS by decomposing the initial failing query into subqueries, starting with those with the lowest number of triples to those with the largest one, and all MFSes are computed gradually. The task is based on the following principle:

Let $\mathcal{D}$ be an RDF database. Assume a failing user query $Q = t_1 \wedge \ldots \wedge t_n$ and $Q_i = t_i$ ($t_i \in \Sigma$, $1 \leq i \leq n$) is a subquery of Q. Clearly, an initial search runs for query evaluation on these initial subqueries Q_i ($1 \leq i \leq n$) (i.e., $[[Q_i]]_{\mathcal{D}} \stackrel{?}{=} \emptyset$) might encounter some actual MFSes. Our method makes use of a sliding approach allowing for an incremental search in the sense that it computes MFSes of increasing size, progressively. In this respect, if the subquery Q_i is failing, then $\{Q_i\}$ is an MFS as it is minimal w.r.t. subset inclusion. Clearly, we do not need to test the proper super-queries of Q_i since, despite these super-queries are guaranteed to fail, they cannot be actual MFSes as they are not minimal w.r.t. set-theoretic inclusion. This progressive increment ensures that the exponential search space is reduced significantly.

Now, if the subquery Q_i succeed, this does not mean that all proper super-queries of Q_i are succeeding on their turn and an exhaustive search is performed. More precisely, the evaluation check is repeated on the subsets $Q_j = Q_i \wedge t_k, t_k \in \Sigma \setminus \{Q_i\}$, successively, until Σ is empty. Then, all MFSes have been found. Once again, when all the remaining MFSes are recorded, our incremental approach is able to exploit this information in a very valuable and efficient way. Algorithm 1, that we call **ALL_MFSes** (Computing All MFSes), sketches out the pseudocode of the approach.

Algorithm 1: ALL_MFSes (Computing All MFSes)

Input: $\mathcal{D}$: an RDF database, Q: a failing query, Σ: the set of triple patterns of Q
Output: $\mathcal{M}$: the set of all MFSes of Q

1 $\mathcal{M} \leftarrow \emptyset$; S $\leftarrow \emptyset$; $\beta \leftarrow \bigcup\{t_i\}, t_i \in \Sigma$;
2 $loop = True$;
3 **while** *(loop = True)* **do**
4 **for** $(\Gamma \in \beta)$ **do**
5 **if** $([[\bigwedge \Gamma]]_{\mathcal{D}} = \emptyset)$ **then**
6 $\mathcal{M} \leftarrow \mathcal{M} \cup \{\Gamma\}$;
7 **else**
8 $S \leftarrow S \cup \{\Gamma\}$;
9 **if** *($S \neq \emptyset$)* **then**
10 $\beta \leftarrow QueryComposition(X, Y)$, $(X, Y) \in S$ with $(X \cup Y) \not\subseteq Z$ $(Z \in \mathcal{M})$, $|X \cap Y| = |X| - 1$;
11 $S \leftarrow \emptyset$;
12 **else**
13 $loop = False$;
14 **return** $\mathcal{M}$;

Algorithm 1 takes as an input an RDF database $\mathcal{D}$, a failing query Q and the set of triple patterns Σ of Q, and outputs the set of MFSes of Q. The algorithm starts with the set of sets β that initially contains the triple patterns of Q (line 1). Then, in each iteration, if the subquery $\bigwedge \Gamma$ fails against $\mathcal{D}$ (line 5), the subset Γ is added to the solution set $\mathcal{M}$ as it is minimal w.r.t. subset inclusion (line 6); otherwise, Algorithm 1 augments S with Γ (line 8). Now, if S is not empty, we call the method **QueryComposition** (line 10) to get the next candidate MFSes as follows: for each pair of distinct sets X and Y from S, Algorithm 1 first checks if X and Y share exactly $|X| - 1$ triple patterns s.t. $(X \cup Y)$ is not contained in any MFS of $\mathcal{M}$ (to prevent the same MFS from being computed again). If that is the case, this implies that $(X \cup Y)$ could be a potential MFS and Algorithm 1 adds $(X \cup Y)$ to β. Then, the evaluation check is repeated on the new subsets of β (lines 4-8). If S is empty, the entire set of MFSes has been found, and the algorithm terminates. Figure 1 shows the working of our algorithm on Example 1.

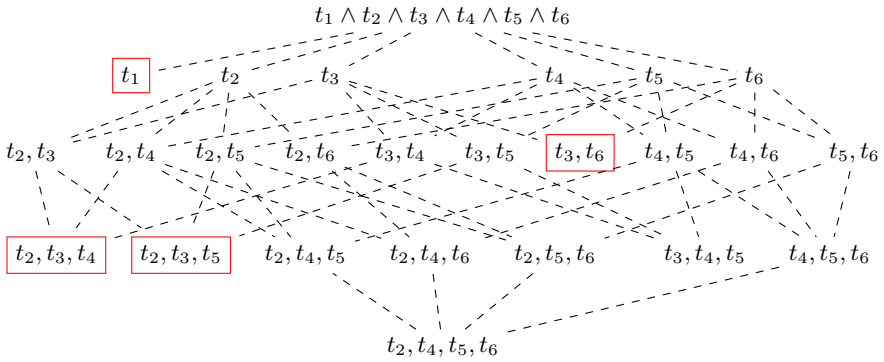


Figure 1: This figure illustrates the search process (Algorithm 1) to compute all the MFSes of Q (Example 1)

3.2 Step 2: Finding all Relaxed Queries

In this step, we present our complete approach for automatically finding all relaxed queries which approximately meet the original query intention. Interestingly, once all MFSes are computed in Step 1, we will show how these MFSes allow us to enumerate the complete set of maximal succeeding subsets without any access to the RDF database. To do so, we start firstly by searching the *hitting sets* of MFSes which correspond to a set-theoretical complement of XSSes w.r.t. a failing RDF query. Formally, we define this notion as follows.

Definition 3 *Let $Q = t_1 \wedge \ldots \wedge t_n$ be a failing RDF query and $\Sigma = \{t_1, \ldots, t_n\}$ be the set of triple patterns of Q. The complement of an XSS Γ of Q, denoted by CoXSS, is defined as $\Sigma \setminus \Gamma$.*

As a shortcut, the set of all complement maximally succeeding subsets of Q will be identified with CoXSSes.

Corollary 1 *Let $Q = t_1 \wedge \ldots \wedge t_n$ be a failing RDF query and $\Sigma = \{t_1, \ldots, t_n\}$ be the set of triple patterns of Q. Then, we have CoXSSes$(Q) = \{\Gamma \subseteq \Sigma \mid (\Sigma \setminus \Gamma) \in XSSes(Q)\}$.*

The next property shows that a CoXSS is a set of triple patterns not included in some minimal failing subsets.

Proposition 1 *Let Q be a failing RDF query. Let X be the set of MFSes of Q. Then, a CoXSS of Q contains at least one triple pattern from each MFS $\Gamma \in X$.*

Proposition 1 is based on the fact that any satisfiable subset of a failing query cannot fully contain any failing subset, and thus this successful subset must exclude at least one triple pattern from every failing set. This relationship will be exploited in the relaxation analysis by using one type of result to guide searches for the other.

The efficiency of our approach relies on the crucial concept of *hitting set*, which appears to be a powerful ingredient of our technique to locate all XSSes of the failing user query. For this, let us firstly introduce the concept of hitting sets.

Definition 4 *Given a collection Ω of sets. A set H is a hitting set for Ω if $H \subseteq \bigcup \Omega$, and $\forall S \in \Omega$, $H \cap S \neq \emptyset$. If no proper subset of H is a hitting set for Ω, then H is a minimal hitting set for Ω.*

Interestingly, the relationship between CoXSSes and the set of MFSes of a failing RDF query stated by Proposition 1 can be described as a solution to a set covering problem. More specifically, the following result shows that our method computes the complete set of CoXSSes from the set of MFSes via the hitting set problem.

Proposition 2 *Let Q be a failing RDF query. Then, a CoXSS of Q is a minimal hitting set of the set of MFSes of Q.*

Intuitively, Proposition 2 shows that the set of minimal failing sets implicitly encodes the entire set of relaxations of the original failing query. The relationship between MFSes and CoXSSes is that they are *hitting set duals* of one another, that is, each CoXSS has at least one triple pattern in common with all MFSes (and is minimal in this sense), and vice versa. More precisely, a hitting set of a collection of sets, is a set that contains at least one element from each set in the collection. In this case, the collection is the set of MFSes, and a CoXSS is a set of triples that contains at least one triple pattern from every MFS. Note that a CoXSS is a hitting set of the MFSes with the additional restriction that it cannot be any smaller without losing its defining property: it is an *irreducible hitting set*.

Accordingly, a CoXSS is an irreducible hitting set of the set of MFSes that represent minimal sets of triples that should be dropped in order to restore consistency of the original query. In a dual way, every MFS of a failing query is also an irreducible intersecting set of CoXSSes. In this way, although the minimal hitting set is a difficult problem [Liffiton and Sakallah, 2005], its irreducibility makes this problem less resource-intensive. So, from this particularity a CoXSS can even be deduced in polynomial time from the set of MFSes.

Now, once the entire set of CoXSSes has been computed, the second phase of Step 2 produces the set of all XSSes of the original failing RDF query, as stated in Corollary 1. More precisely, given a CoXSS Γ of the failing RDF query Q with Σ the set of triple patterns of Q, then the corresponding XSS of Q is $\Gamma' = \Sigma \setminus \Gamma$. Consequently, $\bigwedge \Gamma'$ is a relaxed query of Q. Obviously enough, the number of relaxed queries of Q is equal to the cardinality of the set of CoXSSes. Interestingly, contrary to the existing approaches that query the RDF database at least once in order to find an XSS [Fokou *et al.*, 2017], our computing process of all relaxed queries is done

in a direct way without without querying databases.

In light of Proposition 2, we present our Algorithm 2, called **ALL_RQ** (Computing All Relaxed Queries) to determine the possible relaxed queries. It is based on an original counting heuristic grafted to the Minimal-to-Maximal Conversion Search (MMCS) algorithm [Murakami and Uno, 2014], which explores the set of MFSes in order to compute efficiently the irreducible hitting sets of these MFSes. These irreducible hitting sets are all CoXSSes of the user query. Then, our algorithm computes the complete set of XSSes from the set of CoXSSes in a straightforward manner. Finally, all relaxed queries are generated using the set of XSSes. For instance, let us consider the excluded sets of triple patterns were left out in a set of MFSes with Algorithm 1: MFSes = $\{\{t_1\}, \{t_3, t_6\}, \{t_2, t_3, t_4\}, \{t_2, t_3, t_5\}\}$. At this step, the minimal hitting sets are computed to output the complete set of CoXSSes = $\{\{t_1, t_3\}, \{t_1, t_2, t_6\}, \{t_1, t_4, t_5, t_6\}\}$. For every CoXSS found previously, Algorithm 2 deletes this complement from the initial set of triple patterns Σ and only keeps those that are not included in the corresponding CoXSS. Then, we obtain $\{t_2, t_3\}$, $\{t_3, t_4, t_5\}$ and $\{t_2, t_4, t_5, t_6\}$ which are the set of all XSSes of the initial query. The set of all relaxed queries of the original query Q are then $t_2 \wedge t_3$, $t_3 \wedge t_4 \wedge t_5$, and $t_2 \wedge t_4 \wedge t_5 \wedge t_6$.

Algorithm 2: ALL_RQ (Computing All Relaxed Queries)

Input: $\mathcal{M}$: the set of MFSes of the query Q, Σ: the set of triple patterns of Q
Output: $\mathcal{RQ}$: the set of all relaxed queries of Q

```
RQ ← ∅;
CoXSSes ← MMCS(MFSes);
for Γ ∈ CoXSSes do
    Γ' ← Σ \ Γ;
    RQ ← ⋀Γ' ∪ RQ;
return RQ;
```

Proposition 3 *Algorithm ALL_RQ is sound and complete, i.e., it returns exactly all relaxed queries for a failing user query Q.*

4 Empirical Investigation

In this section, we report different experiments to evaluate the scalability and efficiency of CADER. For baseline comparison, we retain the dedicated algorithms LBA and MBA proposed recently by Fokou et al. [Fokou *et al.*, 2017], which in turn are shown more competitive than all previous approaches by Godfrey [Godfrey, 1997], and the depth-first search algorithm [Fokou *et al.*, 2015].

4.1 Experimental Settings

Software & hardware. CADER has been implemented in Java, with Jena library. For the second step of it, the complete search of all relaxed queries is based on the use of the MMCS algorithm [Murakami and Uno, 2014], which is currently one of the best modern solver that enumerates all minimal hitting sets. All our experiments were performed on a 2Ghz Intel core i7 PC with 16GB of memory, running Ubuntu Linux.

As input of the tested algorithms, we consider different failing RDF queries, compute their MFSes and return the set of all relaxed queries (i.e. XSSes). In the following, a cutoff time has been set to 100 seconds per query. Moreover, missing bars in Figure 3 correspond to executions which timed out. This indicates that no result has been obtained within 100 seconds CPU time. Results are obtained by averaging over 10 runs. All data, queries, results and software used in the experimentations are available from https://github.com/CADER-Project/CADER-Relaxation.git.

Data. We conducted experiments using real DBpedia data [Lehmann *et al.*, 2015]. The DBpedia ontology consists of 320 classes which form a subsumption hierarchy and are described by 1650 different properties. This dataset is the structured counterpart of Wikipedia and it is modeled as an RDF graph. We also used the well-known LUBM dataset [Guo *et al.*, 2005], a synthetic ontology developed to benchmark knowledge base systems w.r.t. large OWL applications. The ontology is situated in the university domain, and is based on 43 classes and 32 properties hierarchies. The background knowledge, i.e. the terminology, is described in a schema called Univ-Bench [Guo *et al.*, 2005]. LUBM is widely used by the Semantic Web community for benchmarking triple-stores. Using the LUBM data generator, we have created test sets containing between 65 millions (LUBM500) and 1 billion (LUBM10k) triples, to obtain synthetic datasets of incremental sizes. The main characteristics of the different datasets are summarized in Table 1.

Queries. To evaluate the query relaxation performance of our system, there is no open domain benchmark query set for RDF empty-answer problem. Therefore, twenty nine queries were constructed over DBpedia. Moreover, for LUBM dataset we consider in our experiment a set of 197 queries generated by [Fokou *et al.*, 2017]. The queries range from 2 to 20 triple patterns. The DBpedia and LUBM queries are of varying shape: (a) *star-shaped*; (b) *chain-shaped*; and (c) *composite-shaped* queries.

Dataset	#Triples	#Instances	Size
DBpedia	1B		
LUBM500	65M	11M	620MB
LUBM 1K	110M	28M	1.29GB
LUBM 3K	...M	...M	...GB
LUBM 5K	550M	140M	2.8GB
LUBM 7K	...M	...M	...GB
LUBM 10K	1B	270M	3.6GB

Table 1: Statistics of datasets used in experiments

4.2 Experimental Results

Various experiments have been carried out. Figure 2 depicts the evaluation time, for our 30 queries, of three RDF relaxation algorithms (i) LBA, (ii) MBA, and (iii) CADER. For each method, the reported runtime is in seconds and it includes the time to compute all the MFSes and the relaxed queries for a given RDF query against DBpedia dataset.

Moreover, Figure 3 summarizes our empirical results by varying the dataset size. In addition, Figure 4 depicts the time spent by CADER to compute MFSes and relaxed queries separately. It should also be noted that the running time reported for the query Q_1 (on all datasets) includes also the time taken by the underlying algorithm for preparing the RDF database prior to answering queries.

Experiments with Star-Shaped Queries

Figure 2 shows the query evaluation times per dataset and system, thus summarizing the results of our experiments on all 36 test star-shaped queries. We represent the computation time to enumerate the set of all MFSes and relaxed queries (i.e. XSSes) w.r.t. the query size. Firstly, we observe in Figure 2 that our approach CADER enumerates the set of all MFSes and relaxed queries for all star-shaped queries in the different datasets, which is not the case for LBA and MBA algorithms. For instance, LBA and MBA were unable to compute all MFSes and XSSes of the queries Q_5 and Q_6 within 100 seconds CPU time. This can be explain by the fact that for larger queries, the number of executed subqueries exponentially increases for LBA and MBA. Further, MBA used a matrix to identify subqueries which let the computation time more important than LBA for some queries, since the size of the matrix can be large for those queries. Thus, the performance of these two baselines quickly decreases for large queries.

Also, as we can see from Figure 2, the running time for LBA and MBA increases for large datasets (e.g., LUBMxk, $x \in \{1, 4, 5, 10\}$). This time can reach 2.5 seconds and 2.3 seconds for LBA and MBA for every query with 10 triple patterns, respectively. Interestingly, as the size of dataset increases, CADER always finishes regardless of the number of the triple patterns. It keeps returning answers for all queries in at most 0.4 seconds. By allowing complete sets of relaxed queries to be delivered in a shorter time from the set of MFSes, CADER is about 12X faster than LBA and MBA for the query Q_4 (on LUBM10k).

We end this subsection by claiming that the intuition of why our approach is clearly more efficient and scalable than the baselines, is the fact that finding relaxed queries can be done in a more efficient way by formulating this problem as a minimal hitting set problem.

Hence, Figure 4 illustrates how the shorter time for computing relaxed queries does not exceed 0.02 seconds.

Experiments with Chain-Shaped Queries

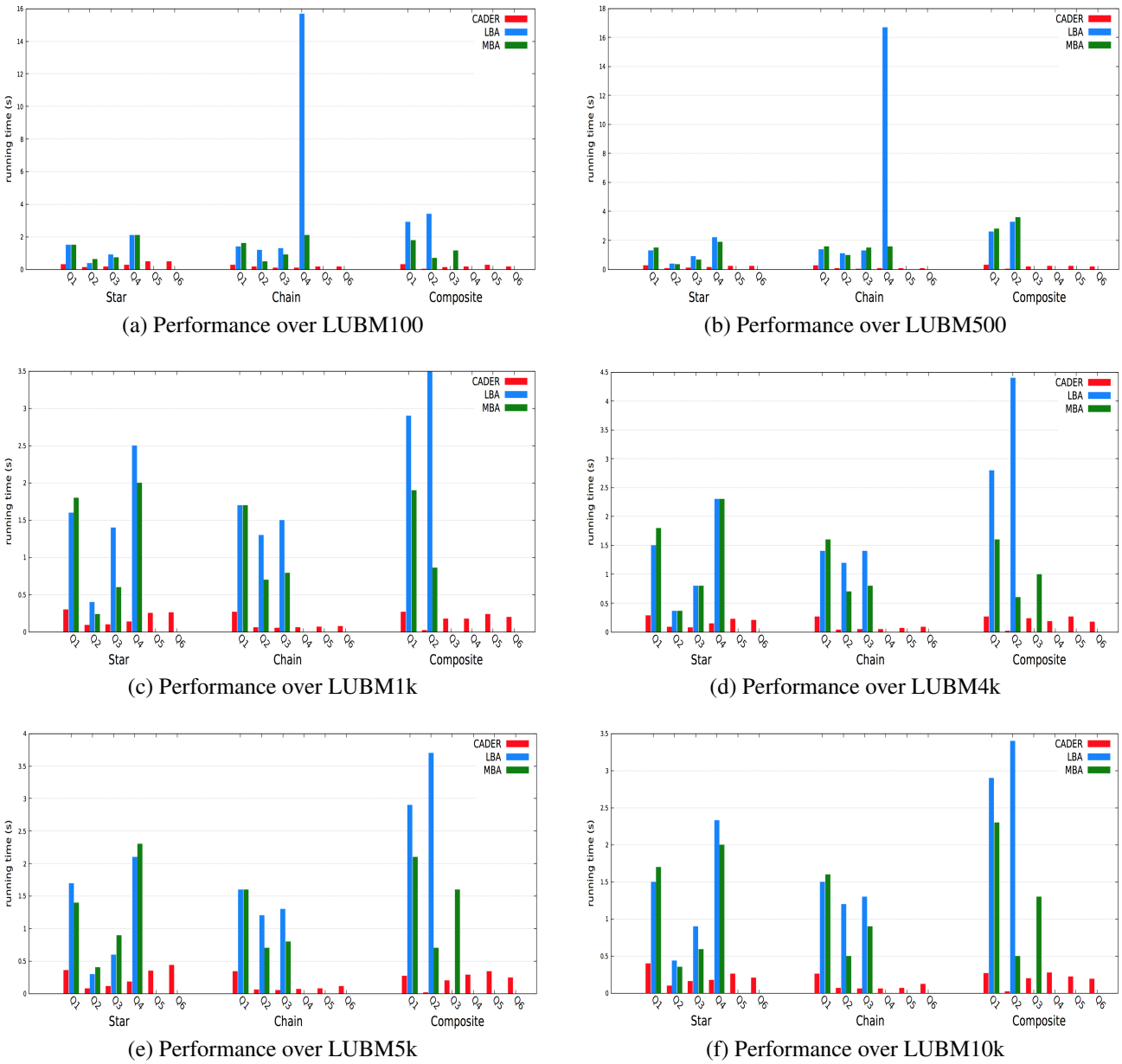
In the next set of experiments, results on chain-shaped queries to compute the complete set of MFSes and the relaxed queries are reported. In the same Figure 2, our approach offers a good trade-off between the number of MFSes and relaxed queries and the time spent. A significant time gap can be observed in favor of CADER. First, the set of MFSes and XSSes of both algorithms LBA and MBA on chain queries over LUBM100 and LUBM500 is generated only for the first four queries as shown in Figure 2 (a, b). When the data size increases (e.g., for LUBMxk, $x \in \{1, 4, 5, 10\}$), LBA and MBA running times become less effective and slower and they scale only for queries with less than 10 triple patterns (see Figure 2 (c-f)). For instance, LBA needs more than 15 seconds (on LUBM100 and LUBM500) to relax the query Q_4. LBA and MBA become inapplicable under the time limit for chain queries with more than 10 triple patterns. This can be explained by the fact that the search space can be very large to compute all MFSes and XSSes for these queries. In contrast, CADER's performance is robust, the best in all cases. Our algorithm is able to scale for all queries and returns all the possible relaxations on the different datasets. More interestingly, a significant time gap can be observed in favor of CADER on chain-shaped queries for LUBMxk ($x \in \{1, 4, 5, 10\}$). In addition, our algorithm discovered all MFSes and relaxed queries for all chain-shaped queries (on all LUBM datasets) within at most one second. This behavior is explained by the fact that CADER exploits earlier information (i.e., MFSes) to enumerate all relaxed queries without further querying the database. Clearly, Figure 4 shows that CADER is able to compute all the relaxed queries in a very short period of time (at most 0.02 seconds), which gave our algorithm huge advantages to scale over large datasets.

Let us also emphasize that even on large instances like LUBMxk ($x \in \{1, 4, 5, 10\}$) our algorithm does not exceed 0.1 seconds to deliver all MFSes and the relaxed queries. Overall, CADER outperforms the other systems by several orders of magnitude on all LUBM datasets.

Experiments with Composite-Shaped Queries

In the last experiment, we further analyzed the scalability of our algorithm on more sophisticated queries, i.e. composite-shaped queries that involve different complex join query patterns and not only object-subject join pattern like with chain-shaped queries. A significant time gap can be clearly observed in favor of CADER. The efficiency gain ratio is even more significant when the size of the dataset increases. Once again, LBA failed to provide results for queries involving more than 5 triple patterns for all the LUBM datasets. We see also that LBA is slower than MBA for the queries Q_1 and Q_2 on all datasets except for LUBM500. Moreover, MBA runs for the query Q_3 on all datasets except for LUBM500 and LUBM1k. As our experimental results illustrate, CADER still being very efficient for queries with large number of triples (e.g. Q_4,Q_5 and Q_6). As the size of dataset increases, CADER shows better performance than LBA and MBA. More interestingly, CADER keeps returning answers for every composite query in at most 0.3 seconds. Besides, Figure 3 shows that CADER outperforms LBA and MBA algorithms for different orders of magnitude. For instance, in the largest dataset (LUBM10k), CADER is 21 (resp. 8) order of magnitude faster than LBA (resp. MBA) for the first two (resp. three) composites queries. An important reason behind this is that CADER can process the computation of MFSes and relaxed queries in less operations, while LBA and MBA need more additional scans of the dataset to determine the set of MFSes and XSSes. This can be explained by the fact that composite queries are the most complicated ones and the evaluation of this type of queries requires additional computation time.

Notice that for CADER, the running time to relax the query Q_6 with 15 triple patterns is lower than for Q_5 (13 triple patterns) on all LUBM datasets even the number of MFSes for



(a) Performance over LUBM100

(b) Performance over LUBM500

(c) Performance over LUBM1k

(d) Performance over LUBM4k

(e) Performance over LUBM5k

(f) Performance over LUBM10k

Figure 2: Query performance comparison for LUBM datasets (Query execution times??)

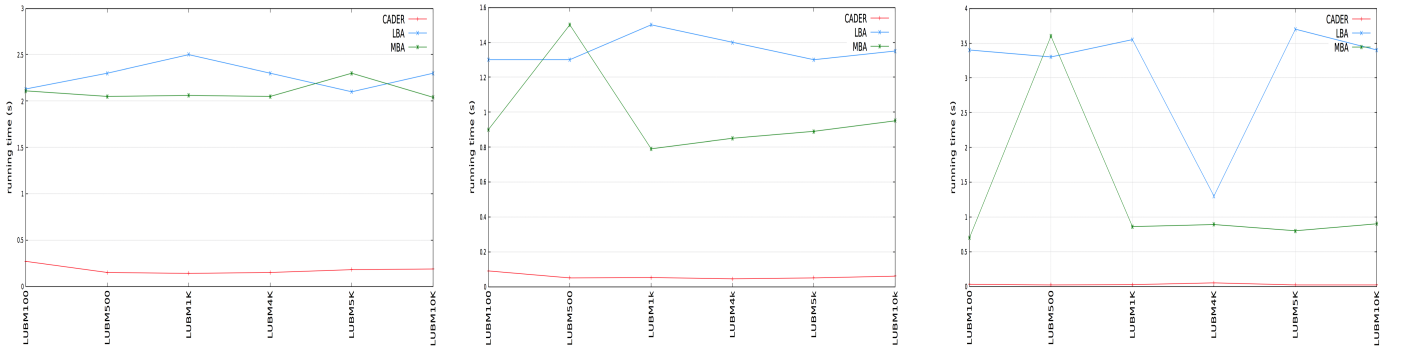
Q_6 is greater than for Q_5. This is due to the fact that the MFSes of Q_6 are found in the first levels of the search process. Finally, let us note that the additional time spent by CADER to compute all relaxed queries from the set MFSes is often very small (0.02 seconds), thanks to the MMCS hitting set algorithm (see Figure 4).

Overall, the observed results in Figure 2 shows clearly that CADER is the fastest algorithm to find all the possible relaxations. Experiments indicate also that the performance gains depend on a number of factors including the size of the RDF dataset, the number of triple patterns of the query and the type of query itself. Hence, the poor performance of LBA and MBA correlates with the large datasets and the shape of the query.

Varying Dataset Size

For more deeper analysis, we have evaluated the scalability of our algorithm when the size of the dataset increases. Figure 3 presents the runtime of the different algorithms for the star Q_4, chain Q_3 and composite Q_2 queries [1], when the dataset ranges from 13M to 1B triples. As can be seen, LBA and MBA are both very much slower than CADER, specially for star and chain queries, since they need to identify the set of XSSes once an MFS is found. These XSSes should not in-

[1] We have chosen the largest query from each type if all the algorithms were able to scale under the time limit.



(a) Performance on the star query Q_4 (b) Performance on the chain query Q_3 (c) Performance on the composite query Q_2

Figure 3: Execution time vs data size

clude the MFS previously computed, and this process is recursively applied by querying the RDF database once another MFS is identified. Also, MBA was slightly faster than LBA, since it uses a relaxed matrix [Fokou *et al.*, 2017] to compute the set of XSSes without the need for database queries. Unfortunately, the computation of such matrix is very expensive. Hence, LBA and MBA can get into trouble for larger datasets. Figure 3 further highlights that CADER is always the best regardless of the shape of RDF queries. Even more interesting is that CADER is very performant for more complex queries (see Figure 3 (c)). The running time is then very small, between 0.029 seconds (on LUBM100) and 0.027 seconds (on LUBM10k).

Number of Database Queries

Let us now consider the number of executed queries. Table 2 shows the number of executed queries for each workload query. The aim is to show how the number of required database queries impacts the performance of the RDF query relaxation process. For this, we have chosen the largest dataset, i.e. LUBM10k. As we can see, impressive improvements could be achieved for all queries. For all queries, it is easy to see that the improvements are very strong (with a number of database queries reduction of over 93%). Overall, the number of database queries for computing MFSes and relaxed queries is drastically reduced. In addition, we can also see that the number of MFSes is always greater than the set of XSSes. In addition, Figure 4 shows clearly that the query evaluation against the underlying database (i.e., MFSes computation) is typically the most costly operation. The time to compute all relaxed queries is very small than that for MFSes.

Evaluating scalability

We evaluate the scalability of query relaxation methods by measuring each method's running time on synthetic datasets as we increase the instance size. Using the LUBM data generator, we generate synthetic datasets with the ... set to 0.36 and 0.32, respectively. Figure [] shows the running time of methods versus the dataset size. Overall, we notice that CADER is the fastest method. CADER has a considerable advantage in scalability, as it is about an ?? order of magnitude faster.

We provide a convenient user interface for relaxing failing queries over any RDF dataset.

In summary, the following conclusions can be drawn from the empirical evaluation:

1. CADER is is consistently faster and able to speed up computation to several orders of magnitude w.r.t. LBA and MBA;
2. CADER is able to handle large RDF queries with different shapes (star, chain, composite);
3. the efficiency gain ratio of CADER is even more significant when the size of the dataset increased (1 billion of triple patterns);
4. the additional time spent by CADER to compute all relaxed queries from the set of MFSes is often very small.

This is due to the fact that the complete sets of relaxed queries are rapidly deduced from the set of MFSes without the need for further database querying As depicted by Figure 4, on average the time spent by CADER to calculate this set is 0.02 seconds.

5 Related Work

In this section we review the closest works related to our proposal done both in the context of relational databases and RDF query relaxation.

These relaxed queries can return generalized or neighbourhood information by relaxing the original query conditions.
Query relaxation. In [Godfrey, 1997], Godfrey was the first author who proposed to look for constraints responsible for the query's failure and introduced the notion of minimal failing subquery in relational databases but with high computational cost considering that the problem of finding all minimal failing subqueries of a failing query is NP-hard. Additionally, Jannach [Jannach, 2009] and McSherry [McSherry, 2005] have applied the same relaxation techniques to the recommender systems domain where the former leverages a particular matrix to find the minimum relaxations, and computes a successful subquery, while the latter exploits a lattice of subqueries by searching the minimal causes of failure of the user query from the lowest number of constraints to the largest ones.

Recently, Muslea and Lee proposed an online approach based on bayesian causal structures discovery to generate

	Query	#Triples	LBA			MBA			CADER		
			#XSS	#MFS	#Ex. queries	#XSS	#MFS	#Ex. queries	$\#\mathcal{RQ}$	#MFS	#Ex. queries
Star	Q_1	3	1	2	90	1	2	90	1	2	6
	Q_2	5	2	4	654	2	4	654	2	4	12
	Q_3	8	2	8	6594	2	8	6594	2	8	32
	Q_4	10	3	9	28272	3	9	28272	3	9	58
	Q_5	13	–	–	–	–	–	–	3	11	110
	Q_6	15	–	–	–	–	–	–	3	17	124
Chain	Q_1	3	1	2	156	1	2	156	1	2	3
	Q_2	5	1	3	540	1	3	570	1	3	8
	Q_3	8	2	5	6948	2	5	6300	2	5	16
	Q_4	10	–	–	–	–	–	–	2	6	41
	Q_5	13	–	–	–	–	–	–	4	7	150
	Q_6	15	–	–	–	–	–	–	4	8	295
Composite	Q_1	3	1	1	102	1	1	54	1	1	4
	Q_2	5	1	2	396	1	2	168	1	2	9
	Q_3	8	–	–	–	1	3	1518	1	3	50
	Q_4	10	–	–	–	–	–	–	2	4	99
	Q_5	13	–	–	–	–	–	–	2	7	166
	Q_6	15	–	–	–	–	–	–	2	9	168

Table 2: Number of executed queries for computing all MFSes and relaxed queries on LUBM10k

non-failing queries that are highly similar to the original failed query [Muslea and Lee, 2005; Muslea, 2004]. In the same vein, several approaches have been proposed to relax queries in RDF graphs by generalizing triple patterns using class and property hierarchies [Hurtado *et al.*, 2008; Calì *et al.*, 2014], similarity measures [Hogan *et al.*, 2012; Elbassuoni *et al.*, 2011; Ferré, 2018], query rewriting [Bursztyn *et al.*, 2015] and also user preferences that are used for getting Top-k answers [Hurtado *et al.*, 2009; Huang *et al.*, 2012]. Other methods such as relaxation operators [Fokou *et al.*, 2014; Calì *et al.*, 2014] and query rewriting rules have been studied in order to perform the relaxation procedure.

Unfortunately, as claimed in [Fokou *et al.*, 2017] the major limitation of these previous works on RDF query relaxation is that the user is still not able to identify neither the root causes of failure of the initial query, nor the possible reasons of the wrong behaviour. Therefore, the user can start by relaxing triple patterns that are not responsible of the main query's failure, which can be a very time-consuming in processing and does not need to be modified in the original query. Hence, a long processing time to effectively relax the triple patterns responsible of the main query's failure will be consumed. While, there is a trade-off between optimality of the relaxation and the response time.

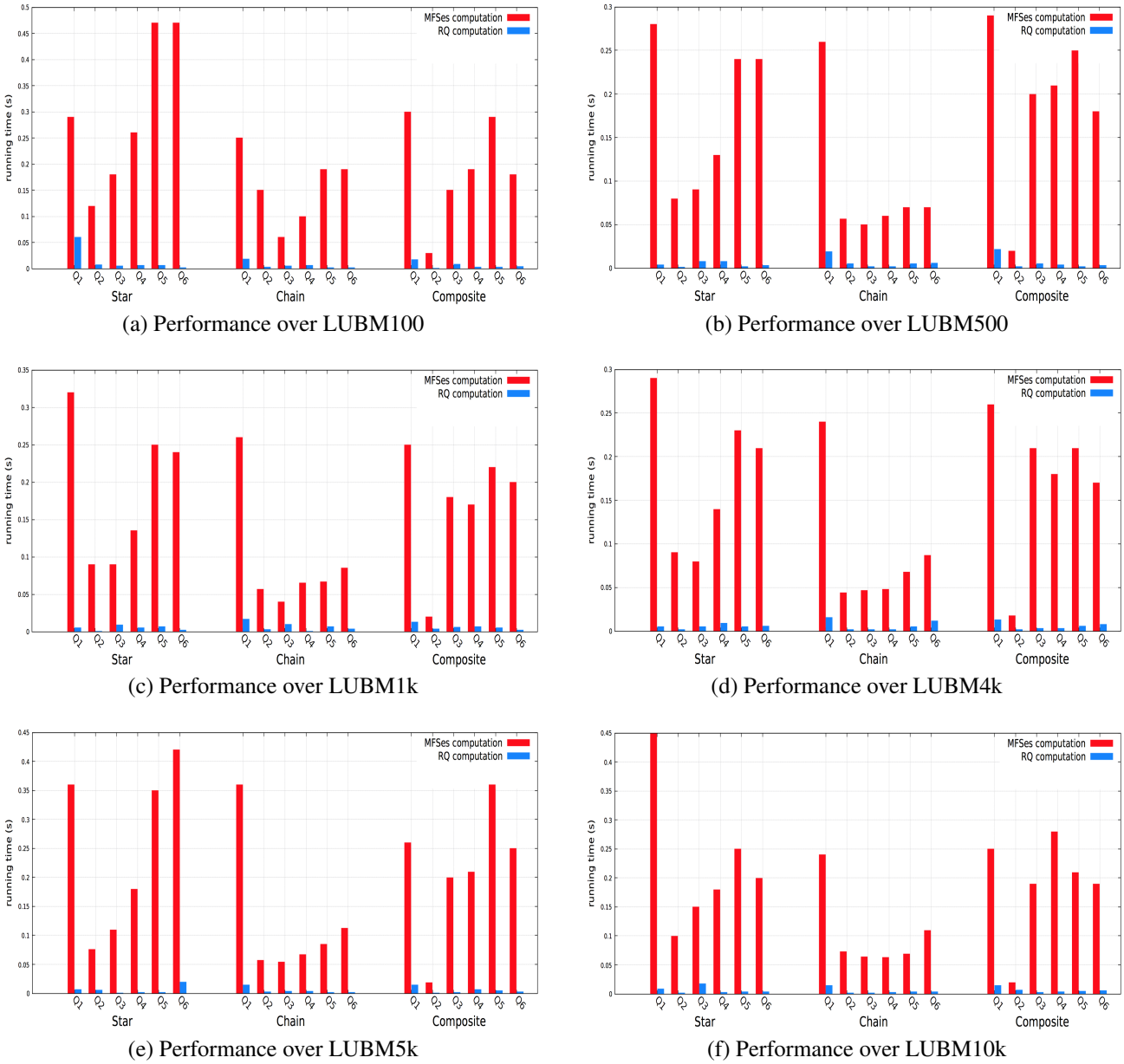
To the best of our knowledge, the only work that studies the issue of computing minimal causes of failure for failing RDF queries is the one proposed by Fokou and al [Fokou *et al.*, 2017]. The authors introduced two algorithmic approaches called *Lattice-Based Approach* (LBA) and *Matrix-Based Approach* (MBA) for the purpose of minimal failing subqueries and maximal succeeding subqueries computation in the RDF context. The former is an adapted and extended variant of Godfrey's ISHMAEL algorithm [Godfrey, 1997], while the latter is inspired by the work of Jannach's method [Jannach, 2009], and is based on a matrix called the relaxed matrix.

In spite of that, the LBA and MBA techniques are not complete in the sense that they do not necessarily deliver all minimal failing and maximal succeeding subqueries for larger queries and become impractical for queries with complicated shape (chain and composite), since they explore an exponential search space. Moreover, the computation of MFSes and XSSes need to query many times the database in a repetitive way.

Moreover, conflicts and relaxations are studied and used in many areas of automated reasoning such as truth maintenance systems (TMS), non-monotonic reasoning, model-based diagnosis, intelligent search, and recently explanations for constraint satisfaction problems (CSPs). In particular in the field of Model-Based Diagnosis, causes of failure are often used as a basis to systematically determine possible explanations, i.e. diagnoses, for an unexpected behavior of the system under observation [Reiter, 1987]. We thus based our contribution on these concepts to find the relaxations of a failing RDF query driven by the causes of failure which can be considered as explanations that may help the user to understand what is wrong in his/her query.

As a result, in this paper, we introduce a new efficient approach to compute all minimal failing and maximal succeeding subsets of a failing RDF query. It improves the current most efficient ones in the literature, namely LBA and MBA [Fokou *et al.*, 2017]. It should be also noted again that our approach is complete, explicable and scalable. In addition, the efficiency of the proposed approach relies on the crucial concept of minimal hitting sets, which appears to be a powerful ingredient of our technique to locate all relaxed queries. Finally, in order to increase the query relaxation efficiency, our technique explores the duality between minimal hitting sets and causes of failures and uses this symmetry to guide the search of all the relaxed queries.

Query reformulation. Query reformulation or Query rewrit-



(a) Performance over LUBM100

(b) Performance over LUBM500

(c) Performance over LUBM1k

(d) Performance over LUBM4k

(e) Performance over LUBM5k

(f) Performance over LUBM10k

Figure 4: MFSes and relaxed queries running times for LUBM datasets for CADER

ing has been an active topic of research spanning relational databases, text search systems, logical databases and now graph databases.

6 Summary & Outlook

Computing all failure causes and query relaxations are highly intractable issues in the worst case. However, it can make sense to compute them for some real-life applications, e.g., user query relaxation in information retrieval. In this paper, we have developed an efficient technique for finding the set of all the possible relaxations of a failing RDF query as well as the set of all the causes of failure that fall to be assessed as explanations. Our approach is based on a strong relationship between failing subqueries and the minimal hitting set problem, so that finding all relaxed queries can be accomplished by finding the minimal hitting set of the MFSes of the original user query. This relationship has the advantage to reduce the number of the RDF database access, which is really very time-consuming and costly. Experimental results show that our approach clearly outperforms drastically the state-of-the-art techniques for RDF query relaxation on large datasets, e.g., LUBM5k and LUBM10k. Moreover, CADER scales up well for queries of 15 triple patterns, while the computation process studied in the related work can be stopped at any time while still having issues to scale with larger queries.

In the future, we plan to study how possible query relax-

ations could be computed progressively from the growing set of extracted MFSes. An optimization of Algorithm 1 to compute MFSes more efficiency is part of our future work. We shall also investigate the applicability of our approach and tackle the problem of query relaxation on RDF (uncertain) data streams for efficient processing. Finally, we plan to investigate the use of parallel hitting set algorithms [Jannach *et al.*, 2015] to handle very large RDF datasets with large queries.

References

[Belleau *et al.*, 2008] François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5):706–716, 2008.

[Bizer *et al.*, 2009] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - A crystallization point for the web of data. *J. Web Sem.*, 7(3):154–165, 2009.

[Boutet *et al.*, 2007] Emmanuel Boutet, Damien Lieberherr, Michael Tognolli, Michel Schneider, and Amos Bairoch. Uniprotkb/swiss-prot: the manually annotated section of the uniprot knowledgebase. *Plant bioinformatics: methods and protocols*, 41(5):89–112, 2007.

[Bursztyn *et al.*, 2015] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. Optimizing reformulation-based query answering in RDF. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 265–276, 2015.

[Calì *et al.*, 2014] Andrea Calì, Riccardo Frosini, Alexandra Poulovassilis, and Peter T. Wood. Flexible querying for SPARQL. In *CoopIS, and ODBASE*, pages 473–490, 2014.

[Dong *et al.*, 2014] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *International Conference on Knowledge Discovery and Data Mining*, pages 601–610, 2014.

[Elbassuoni *et al.*, 2011] Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. Query relaxation for entity-relationship search. In *The Semanic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC*, pages 62–76, 2011.

[Ferré, 2018] Sébastien Ferré. Answers partitioning and lazy joins for efficient query relaxation and application to similarity search. In *European Semantic Web Conference on Semantic Web ESWC*, pages 209–224, 2018.

[Fokou *et al.*, 2014] Géraud Fokou, Stéphane Jean, and Allel Hadjali. Endowing semantic query languages with advanced relaxation capabilities. In *Foundations of Intelligent Systems - 21st International Symposium, ISMIS*, pages 512–517, 2014.

[Fokou *et al.*, 2015] Géraud Fokou, Stéphane Jean, Allel Hadjali, and Mickaël Baron. Cooperative techniques for SPARQL query relaxation in RDF databases. In *European Semantic Web Conference*, pages 237–252, 2015.

[Fokou *et al.*, 2017] Géraud Fokou, Stéphane Jean, Allel Hadjali, and Mickaël Baron. Handling failing RDF queries: from diagnosis to relaxation. *Knowl. Inf. Syst.*, 50(1):167–195, 2017.

[Godfrey, 1997] Parke Godfrey. Minimization in cooperative response to failing database queries. *Int. J. Cooperative Inf. Syst.*, 6(2):95–149, 1997.

[Guo *et al.*, 2005] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[Hoffart *et al.*, 2013] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[Hogan *et al.*, 2012] Aidan Hogan, Marc Mellotte, Gavin Powell, and Dafni Stampouli. Towards fuzzy query-relaxation for RDF. In *Extended Semantic Web Conference on the Semantic Web: Research and Applications*, pages 687–702, 2012.

[Huang *et al.*, 2012] Hai Huang, Chengfei Liu, and Xiaofang Zhou. Approximating query answering on RDF databases. *World Wide Web*, 15(1):89–114, 2012.

[Hurtado *et al.*, 2008] Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood. Query relaxation in RDF. *J. Data Semantics*, 10:31–61, 2008.

[Hurtado *et al.*, 2009] Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood. Ranking approximate answers to semantic web queries. In *European Semantic Web Conference on the Semantic Web: Research and Applications*, pages 263–277, 2009.

[Jannach *et al.*, 2015] Dietmar Jannach, Thomas Schmitz, and Kostyantyn M. Shchekotykhin. Parallelized hitting set computation for model-based diagnosis. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1503–1510, 2015.

[Jannach, 2009] Dietmar Jannach. Fast computation of query relaxations for knowledge-based recommenders. *AI Commun.*, 22(4):235–248, 2009.

[Lanti *et al.*, 2015] Davide Lanti, Martín Rezk, Guohui Xiao, and Diego Calvanese. The NPD benchmark: Reality check for OBDA systems. In *International Conference on Extending Database Technology EDBT*, pages 617–628, 2015.

[Lehmann *et al.*, 2015] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia -

A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[Liffiton and Sakallah, 2005] Mark H. Liffiton and Karem A. Sakallah. On finding all minimally unsatisfiable subformulas. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 173–186, 2005.

[McSherry, 2005] David McSherry. Retrieval failure and recovery in recommender systems. *Artif. Intell. Rev.*, 24(3-4):319–338, 2005.

[Murakami and Uno, 2014] Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics*, 170:83–94, 2014.

[Muslea and Lee, 2005] Ion Muslea and Thomas J. Lee. Online query relaxation via bayesian causal structures discovery. In *Twentieth National Conference on Artificial Intelligence*, pages 831–836, 2005.

[Muslea, 2004] Ion Muslea. Machine learning for online query relaxation. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 246–255, 2004.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[Stefanoni *et al.*, 2018] Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. Estimating the cardinality of conjunctive queries over RDF data using graph summarisation. In *World Wide Web Conference on World Wide Web WWW*, pages 1043–1052, 2018.