On Relaxing Failing Queries over RDF Databases

Paper ID: #5175

Abstract

In this paper, we present CADER, a novel approach 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 failed subqueries of the user query. Then, we point out how the hitting set problem can be applied for determining the possible relaxed queries in an efficient way without querying databases. Finally, the efficiency and scalability of CADER are shown through extensive experiments on the well-known RDF benchmarks with a variety of queries of different shapes.

1 Introduction

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 triplestores), 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], etc. and they shape a very large set of graphs having millions of triples interlinked each other. Unfortunately, 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, and time-consuming process. To tackle this issue, the following requirements must then be fulfilled:

- **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.

Automated query relaxation algorithms have been studied in the context of relational databases [Godfrey, 1997; Jannach, 2009; McSherry, 2005; Muslea and Lee, 2005; Muslea, 2004] and RDF data [Hurtado *et al.*, 2008; Calì *et al.*, 2014; Hogan *et al.*, 2012; Elbassuoni *et al.*, 2011; Ferré, 2018; Wang *et al.*, 2018]. 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 minimal failing subqueries and maximal succeeding subqueries computation in the RDF context [Fokou *et al.*, 2017]. 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. 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 and explicable to some extent as no answer is given for queries beyond 15 triple patterns in a reasonable response time. 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 is extremely linked to the computation of the root causes of failure in the sense 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 subqueries against the entire 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 DBpedia and LUBM datasets with various sets of queries to show the scalability and efficiency of our approach.

These contributions are the basis of our approach named CADER 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 (23 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. Due to space constraints, the proofs can be found in goo.gl/1zhyS9.

2 Preliminaries

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 super-query* 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 $\mu : V \to U \cup B \cup L$. 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: $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \ and\ \mu_1 \sim \mu_2\}$. Last, given an RDF database $\mathcal{D}$ and a query Q, the evaluation of Q over $\mathcal{D}$, denoted by $[[Q]]_{\mathcal{D}}$, is recursively defined as:

- 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 triple patterns *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 of 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 *Let us consider the database inspired by the LUBM Benchmark [Guo* et al.*, 2005]. Let us consider the following query $Q = t_1 \wedge t_2 \wedge t_3 \wedge t_4 \wedge t_5 \wedge t_6$:*

$$
\begin{array}{ll}
SELECT * WHERE \{ & \\
?Y10\ hasAlumnus\ ?Y11\ \ . & (t_1) \\
?Y11\ advisor\ ?Y12\ \ . & (t_2) \\
?Y12\ advisor\ ?Y13\ \ . & (t_3) \\
?Y13\ doctoralDegreeFrom\ ?Y14\ \ . & (t_4) \\
?Y14\ hasAlumnus\ ?Y15\ \ . & (t_5) \\
?Y15\ title\ 'Dr'\} & (t_6)
\end{array}
$$

This query fails against LUBM since the sets $\{t_1\}$, $\{t_3, t_6\}$, $\{t_2, t_3, t_4\}$, and $\{t_2, t_3, t_5\}\}$ are MFSes of Q. 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**D**F 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.

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, it 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., $[[Q_i]]_{\mathcal{D}} \overset{?}{=} \emptyset$) 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. 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. In other words, 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, $\Sigma$: 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 $(X \cup Y) \not\supseteq Z$ $(Z \in \mathcal{M})$, $|X \cap Y| = |X| - 1$;
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. $(X \cup Y)$ does not contain 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). Finally, 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. All MFSes are depicted with red color and the set of proper super-queries of evry MFS is omitted.

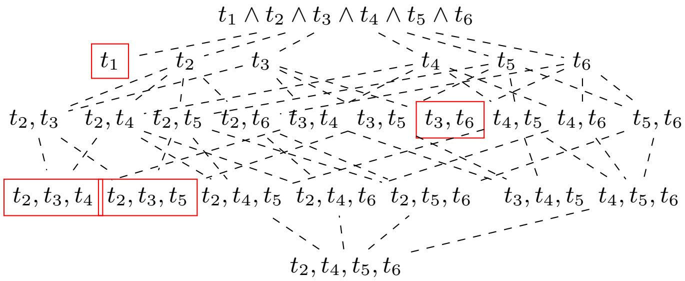

Figure 1: This figure illustrates the search process in Algorithm 1 to compute all the MFSes of Q (Example 1)

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. Then, 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 succeeding 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 triple patterns 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 triple patterns 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 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 set of MFSes returned by 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 $Q$, $\Sigma$: the set
       of triple patterns of $Q$
Output: $\mathcal{RQ}$: the set of all relaxed queries of $Q$
$\mathcal{RQ} \leftarrow \emptyset$;
CoXSSes ← MMCS(MFSes);
for $\Gamma \in$ CoXSSes do
    $\Gamma' \leftarrow \Sigma \setminus \Gamma$;
    $\mathcal{RQ} \leftarrow \bigwedge \Gamma' \cup \mathcal{RQ}$;
return $\mathcal{RQ}$;
```

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 efficiency and scalability of CADER. For baseline comparison, we retain the dedicated algorithms LBA and MBA [Fokou *et al.*, 2017], which in turn are shown more competitive than all existing approaches by [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. Tests were performed on a 2Ghz Intel core i7 PC with 16GB of memory, and running Ubuntu Linux 16.04.

Data. We conducted experiments using *real* and *synthetic* datasets. The DBpedia [Lehmann *et al.*, 2015] dataset consists of real data gathered from Wikipedia. The DBpedia ontology consists of 320 classes which form a subsumption hierarchy and are described by 1650 different properties. 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. LUBM is widely used by the Semantic Web community to compare performance especially when arbitrarily large datasets are required. Using the LUBM data generator [Guo *et al.*, 2005], we have created test sets ranging from 65 millions (LUBM500) to 2 billion (LUBM15k) triples, to obtain 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, we have generated twenty one queries of varying shape over the DBpedia dataset: $Q1$-$Q7$ star-shaped, $Q8$-$Q15$ chain-shaped, and $Q16$-$Q21$ composite-shaped queries. We also borrowed from [Fokou *et al.*, 2017] a set of 197 LUBM queries, having between 2 and 15 triple patterns.

Dataset	#Triples	#Instances
DBpedia	23B	4.2M
LUBM500	65M	11M
LUBM 1K	110M	28M
LUBM 3K	320M	90M
LUBM 5K	550M	140M
LUBM 7K	720M	195M
LUBM 10K	1.3B	270M
LUBM 15K	2B	360M

Table 1: Statistics of datasets used in experiments

4.2 Experimental Results

Various experiments have been carried out and all results are obtained by averaging over 5 runs. Queries whose evaluation requires more than 10 seconds were interrupted. All data, queries, results and a user interface are available from goo.gl/1fZ8us. The graph in Figure 2 shows, for each DBpedia query the number of triple patterns in the query (in parentheses after the query name). For each method, the reported runtime is in milliseconds and it includes the time to compute all the MFSes and the relaxed queries [1] for each RDF query against DBpedia dataset. First, for all methods, the query relaxation response time generally increases as the number of triple patterns grows, except for $Q1$. In fact, the running time for $Q1$ includes also the time spent by the algorithm to initialize the connection to the database. The same figure also shows that our algorithm is able to scale for all queries and returns all the possible relaxations in a very short period of time (at most 195 ms), which is not the case for LBA and MBA algorithms. For instance, the baselines were unable to compute all MFSes and XSSes of the query $Q15$ within 10 seconds CPU time. Further, as we can see from Figure 2, the running time for LBA and MBA increases for some queries with large number of triple patterns. This time can reach 8.4 seconds and 1.7 seconds for baselines for $Q7$ and $Q12$, respectively. This can be explained by the fact that for larger queries, the number of executed subqueries exponentially increases for LBA and MBA. Thus, the performance of these two baselines quickly decreases for these queries. Also, MBA used a matrix to identify subqueries of the original query which let the computation time more important than LBA for some queries (e.g., $Q6$), since the size of the matrix can be large for those queries. Moreover, as can be seen, LBA and MBA are both very much slower than CADER, specially for queries with large number of MFSes (e.g. $Q6$ with 62 MFSes, and $Q7$ with 78 MFSes), since they need to identify the set of XSSes once an MFS is found. These XSSes should not include the MFS previously computed, and this process is recursively applied by querying the RDF database once another MFS is found. Overall, Figure 2 further highlights that CADER is always the best regardless of the shape of DBpedia queries. Interestingly, by allowing complete sets of relaxed queries to be

[1] In this section, we use the terms XSS and relaxed query interchangeably.

delivered in a shorter time from the set of MFSes, CADER is about 70X, 37X, 28X and 32X faster than baselines for the queries $Q9$, $Q10$ $Q11$ and $Q17$, respectively. This confirms the important performance improvement brought by our system to query relaxation in RDF.

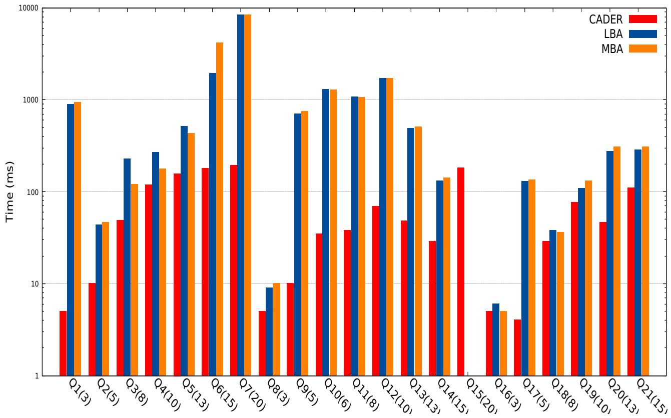

Figure 2: Query relaxation times for DBpedia queries

In addition, Figure 3 depicts the time spent by CADER to compute MFSes and relaxed queries separately. This figure demonstrates that the additional time to compute all relaxed queries from the set MFSes is often very small (at most 14 ms), thanks to the MMCS hitting set algorithm. This shows clearly that the query evaluation against the underlying database is typically the most costly operation.

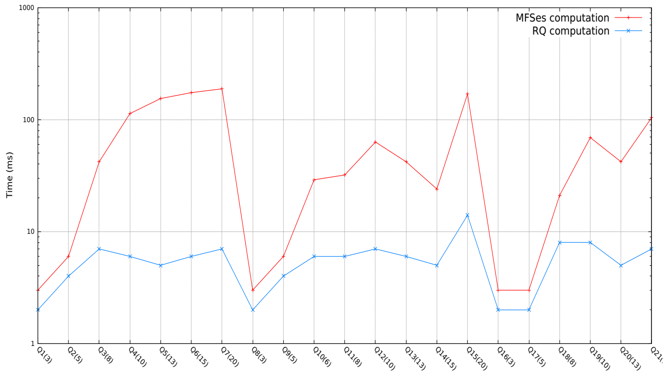

Figure 3: MFSes and RQ running times for DBpedia queries

Evaluating scalability. For more deeper analysis, in the next set of experiments, we evaluate the scalability of query relaxation methods by measuring each method's running time on LUBM datasets as we increase the instance size. Figure 4 presents the average runtime of the different algorithms for the 197 LUBM queries (69 star-shaped, 65 chain-shaped, and 63 composite-shaped queries), when the dataset ranges from 65M to 2B triples. Clearly, a significant time gap can be observed in favor of CADER. The efficiency gain ratio is even more significant when the size of the dataset increases. More precisely, for LUBMxk ($x \in \{7, 10, 15\}$), LBA and MBA running times become less effective and slower, as they need more than 100 ms to relax all the queries. In contrast, CADER's performance is robust, the best in all cases. The running time is then very small, between 30 ms (on LUBM500) and 36 ms (on LUBM15k). Overall, CADER has a considerable advantage in scalability, as it is about 3 order of magnitude faster than baselines. 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. However, LBA and MBA need more additional scans of the dataset to determine the set of MFSes and XSSes. Moreover, this issue can be interpreted by the fact that the LUBM composite queries are the most complicated ones (as they involve different complex join query patterns and not only object-subject join pattern like with chain-shaped queries) and the evaluation of this type of queries requires additional computation time. Hence, LBA and MBA can get into trouble for larger datasets.

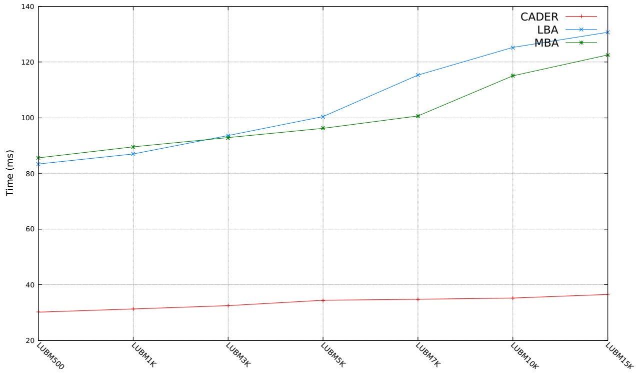

Figure 4: Execution time vs LUBM data size

We end this section 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. 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.

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;

4. the additional time spent by CADER to compute all relaxed queries from the set of MFSes is often very small.

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.

[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 *KDD*, 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 *ESWC*, pages 209–224, 2018.

[Fokou *et al.*, 2015] Géraud Fokou, Stéphane Jean, Allel Hadjali, and Mickaël Baron. Cooperative techniques for SPARQL query relaxation in RDF databases. In *ESWC*, 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 *ESWC*, pages 687–702, 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.

[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 *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 *SAT*, 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 *AAAI*, pages 831–836, 2005.

[Muslea, 2004] Ion Muslea. Machine learning for online query relaxation. In *KDD*, pages 246–255, 2004.

[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 *WWW*, pages 1043–1052, 2018.

[Wang *et al.*, 2018] Meng Wang, Ruijie Wang, Jun Liu, Yihe Chen, Lei Zhang, and Guilin Qi. Towards empty answers in SPARQL: approximating querying with RDF embedding. In *ISWC*, pages 513–529, 2018.