A Survey on SPARQL Query Relaxation under the Lens of RDF Reification

Ginwa Fakih * and Patricia Serrano-Alvarado
LS2N,UMR 6004,Nantes Université, F-44000 Nantes France
E-mails: ginwa.fakih@univ-nantes.fr, Patricia.Serrano-Alvarado@univ-nantes.fr

Abstract. SPARQL query relaxation has been used to cope with the problem of queries that produce none or insufficient answers. The goal is to modify these queries to be able to produce alternative results close to those expected in the original query. Existing approaches generally relax the query constraints based on logical relaxations through RDFS entailment and RDFS ontologies. Techniques also exist that use the similarity of instances based on resource descriptions. These relaxation approaches defined for SPARQL queries over RDF triples have proved their efficiency. Nevertheless, significant challenges arise for query relaxation techniques in the presence of statement-level annotations, i.e., reification. In this survey, we overview query relaxation works with a particular focus on issues and challenges posed by representative reification models, namely, standard reification, named graphs, n-ary relations, singleton properties, and RDF-Star.

Keywords: SPARQL query relaxation, statement-level annotations, reification models, query similarity, query ranking, ontology-based relaxation, similarity of instances

1. Introduction

When a query evaluated over a dataset produces empty or insufficient answers, query issuers may try to modify the query constraints. This task is time-consuming and requires users with a profound knowledge of the data distribution and the schema of the dataset. An efficient way to cope with this problem was proposed in the domain of cooperative answering for deductive databases [1]. The original idea, is a relaxation method to expand the scope of the query dynamically by relaxing the constraints in the query. The goal is to generalize the user query to produce more answers.

In this survey, we focus on SPARQL[1] queries over RDF[2] triples. SPARQL allows defining queries with triple patterns over which conjunctions, disjunctions, and optional patterns can be defined. Query issuers can use the OPTIONAL clause of SPARQL to specify which triple patterns can be ignored if they cannot be satisfied during the query evaluation. This idea is interesting but used to a limited extent because other forms of query relaxation can be used rather than simply dropping triple patterns defined as optionals. In particular, SPARQL query relaxation can be based on a logical relaxation of some of the query constraints by using RDFS entailment and RDFS ontologies. The major challenge of this approach, is that the number of relaxed queries grows combinatorially with the number of relaxation steps and the query size. Several approaches have been proposed to optimize the query relaxation task and generate relaxed queries efficiently. Mainly, the idea is to organize relaxed queries from the most specific to the most general and execute them in this order until obtaining *k-relevant* answers. But many relaxed queries

*Corresponding author. E-mail: ginwa.fakih@univ-nantes.fr.
[1]SPARQL 1.1 Query Language W3C Recommendation https://www.w3.org/TR/sparql11-query/
[2]https://www.w3.org/TR/rdf11-primer/

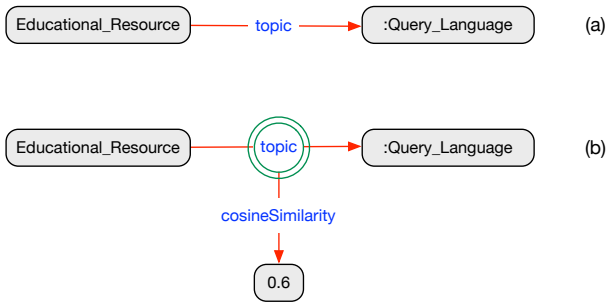


Fig. 1. Representation of context on labeled edge

may produce no new answers because they have the same answers than more specific queries. To cope with this problem, the query execution ordering can be based on the *similarity of queries*. This similarity can be computed using *information content* which is based on the number of instances per class or property. Therefore the challenge is to identify the most similar relaxed queries that may produce new answers and to reduce the space of the relaxed queries that are executed until obtaining k answers.

Other query relaxation approaches use the similarity of instances based on resource descriptions. For example, similarity can be measured using an appropriate function that returns the distance between two attribute values. Such distance can be calculated differently, e.g., lexical similarities (like Jaccard similarity, Jensen-Shannon divergence, cosine similarity), overlap measures, page rank scores, etc. The similarity functions are data-dependent. Thus, the necessary number of functions depends on the different data types. Calculating the similarity of instances can be very costly. Further, multi-valued predicates (triples having a subject-predicate pair with several objects) may induce additional distances to compute. Frequently, the similarity of instances is done offline before the query processing. That is because the distances to compute can be significant, and calculating them dynamically during query processing can be unrealistic.

Moreover, statements about statements, also called statement-level annotations, are increasingly used. They allow specifying that a fact is true under a particular context. Context can concern temporal aspects, provenance, scores, weights, etc. *Reification* allows making statements about statements in a generic manner. For example, Figure 1 illustrates (a) a traditional triple that consists of two resources (nodes) related by a property relation (edge) and (b) a reified triple stating that Query_Language is a topic of an Educational_Resource "to some extent with a cosine similarity of 0.6". In a graph, reification can be seen as defining edges about edges. Reification can be done using several syntaxes that we call models. For example, there is standard reification, named graphs, singleton properties, RDF-star, etc.

Existing query relaxation approaches have proved their efficiency but reification can lead to significant challenges for relaxation techniques.

- If relaxation is applied to a query with a triple pattern like (?er, :topic, :Query_Language), if Query_Language is a class, a *type relaxation* would replace the Query_Language with its superclass Programming_Language without considering the annotated score. So, educational resources with lower scores would be obtained.
- The same drawback would be if a similarity of instances is used, i.e., Language could replace Query_Language even if its cosine similarity would be low.
- Another relaxation for this query could be a *simple relaxation* that would replace Query_Language with a variable, if it is an instance or a constant. This will lead to a huge number of results, e.g., all educational resources regardless of the topic.
- As there is still no recommended way of doing reification, the syntax of SPARQL queries over reified triples can vary significantly. The shape of queries depends on the used reification model. Thus, existing query relaxation approaches will behave differently depending on the reification model.
- Relaxation can also cause performance issues because of existing triplestores' different implementations of reification models. In addition, not all reification models are currently widely supported.

These challenges motivate this survey on query relaxation works with a particular focus on issues and challenges over reification. The goal of this survey is to give a bird's-eye view of the SPARQL queries relaxation approaches over RDF datasets, and compare these approaches based on various relevant criteria.

This paper is structured as follows. Section 2 presents the survey methodology. Section 3 gives preliminary definitions related to query relaxation. Section 4 reviews query relaxation strategies. Section 5 compares and analyses relaxation techniques according to several aspects (query shapes, relaxed terms, optimisation strategies, experimental evaluations, etc.). Section 6 overviews and compares different reification models. Finally, Section 7 discusses the main issues and challenges of applying relaxation on SPARQL queries executed over reified triples. Section 8 concludes.

2. Survey methodology

To the best of our knowledge, there is no survey on SPARQL query relaxation mechanisms. In this section, we present the methodology adopted to select the publications discussed in this survey. The collection of relevant works were found using Google Scholar[3], DBLP[4], HAL[5], ACM Computing Surveys[6], the Semantic Web Journal[7], Springer[8], and Journal of Web Semantics[9]. Most important keywords were: query relaxation, SPARQL queries, SPARQL query relaxation, expanding SPARQL queries, or flexible SPARQL queries.

In our paper collection process, we first screened papers based on their title, abstract, and conclusion. Then this selection was followed by another filtering based on their content. Publications cited in the state-of-the-art sections of relevant publications were also collected and analysed.

We collected papers according to these criteria:

- Papers written in English.
- Papers subjected to peer review, including published journal papers, conference proceedings, book chapters, and workshops.
- Papers published from 2006 and until February 2023. SPARQL became a W3C Candidate Recommendation on 6 April 2006. This paper was submitted in March 2023.

A list of venues representative of the retained papers includes the International Semantic Web Conference (ISWC), the European Semantic Web Conference (ESWC), the World Wide Web Conference (WWW), and the international conference on Web Information Systems Engineering (WISE). In addition, we focused our literature on the papers that study query relaxation using SPARQL. Hence, papers that did not address SPARQL queries were excluded, such as [14–16]. Moreover, the work proposed in [17] is not included in this survey since its contribution is based on successive interactions from the user, whereas, in this survey, we address only automatic methods that do not include any interference from the user. Thus, we review 13 approaches proposed in 12 research papers (see Table 1).

3. Background

Query relaxation is the task of modifying the query conditions automatically when a query fails to generate sufficient solutions. This task helps users generate relaxed queries that return results *close* to those expected in the original query. In the context of RDF, existing approaches generate relaxed queries using different techniques

[3]https://scholar.google.com
[4]https://dblp.org
[5]https://hal.archives-ouvertes.fr/
[6]https://dl.acm.org/journal/csur
[7]https://www.semantic-web-journal.net/
[8]https://link.springer.com/
[9]http://www.websemanticsjournal.org

Year	Reference	Title	Journal/Conference	Authors
2006	[2]	A relaxed approach to RDF querying	ISWC	Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood
2008	[3]	Query relaxation in RDF	Journal on Data Semantics	Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood
2008	[4]	Computing relaxed answers on RDF databases	WISE	Hai Huang, Chengfei Liu, and Xiaofang Zhou
2010	[5]	Combining approximation and relaxation in semantic web path queries	ISWC	Alexandra Poulovassilis, and Peter T. Wood
2010	[6]	Query relaxation for star queries on RDF	WISE	Hai Huang and Chengfei Liu
2011	[7]	Query relaxation for entity-relationship search	ESWC	Shady Elbassouni, Maya Ramanath, and Gerhard Weikum
2012	[8]	Approximating query answering on RDF databases	WWW	Hai Huang, Chengfei Liu, and Xiaofang Zhou
2016	[9]	Towards fuzzy query relaxation for RDF	ESWC	Aidan Hogan, Marc Mellotte, Gavin Powell, and Dafni Stampouli
2016	[10]	RDF query relaxation strategies based on failure causes	ESWC	Geraud Fokou, Stephane Jean, Allel Hadjali, and Mickael Baron
2018	[11]	Answers partitioning and lazy joins for efficient query relaxation and application to similarity search	ESWC	Sebastian Ferre
2019	[12]	On relaxing failing queries over RDF databases	International Conference on Big Data	Wafaa Mebrek, Badran Raddaoui, and Mohamad Albilani.
2021	[13]	Ensuring license compliance in linked data with query relaxation	TLDKS	Benjamin Moreau and Patricia Serrano-Alvarado

Table 1

Table of all the reviewed papers in this survey.

based on logical relaxation using ontologies and RDFS (RDF Semantics) entailment, but also based on similarity of instances using a variety of similarity methods (e.g., statistical language models, probabilistic models, etc.).

There exist plenty of possibilities to relax a query. This depends on the number of elements in every triple pattern that can be relaxed and the hierarchy depth of the ontology. The background presented in this section concerns a review of RDF entailment rules in Section 3.1, an introduction to SPARQL query relaxation in Section 3.2, the similarity of queries based on *information content* in Section 3.3, and the relaxation based on the similarity of instances in Section 3.4.

3.1. RDFS entailment rules

Query relaxation is frequently based on the entailment rules of RDFS[10] that we show in Table 2.

- **Subproperty rules** state that:
  Rule (1) if property *a* is subproperty of property *b*, and b is subproperty of *c*, then it is entailed that *a* is subproperty of *c* ;
  Rule (2) if property *a* is subproperty of *b*, and there exists a triple whose subject is *x*, predicate is *a*, and object is *y*, then the triple with subject *x*, predicate *b* and object *y* is entailed.
- **Subclass rules** state that:
  Rule (3) if class *a* is subclass of *b* that is subclass of *c*, then it is entailed that class *a* is subclass of *c*;
  Rule (4) if class *a* is subclass of *b*, and x is of type *a*, then x is of type *b* too.
- **Typing rules** are based on the domain and range of predicates.
  Rule (5) if property *a* has domain *c*, and there exists a triple whose subject is *x*, predicate is *a*, and object is *y*,

[10] https://www.w3.org/TR/rdf-mt/#RDFSRules

Rule	Statement	Entailment
(1) rdfs5	(a, subProperty, b) AND (b, subProperty, c)	⇒ (a, subProperty, c)
(2) rdfs7	(a, subProperty, b) AND (x, a, y)	⇒ (x, b, y)
(3) rdfs11	(a, subClass, b) AND (b, subClass, c)	⇒ (a, subClass, c)
(4) rdfs9	(a, subClass, b) AND (x, type, a)	⇒ (x, type, b)
(5) rdfs2	(a, domain, c) AND (x, a, y)	⇒ (x, type, c)
(6) rdfs3	(a, range, d) AND (x, a, y)	⇒ (y, type, d)

Table 2
Subproperty, subclass and type entailment rules of RDFS.

then it is entailed that x has type *c*.
Rule (6) if property *a* has range *d*, and there exists a triple whose subject is *x*, predicate is *a*, and object is *y*, then it is entailed that y has type *d*.

3.2. SPARQL query relaxation

The query relaxation explained in this section was proposed in [2] and is largely adopted by approaches using the RDFS entailment rules. We begin this section by introducing the triple pattern relaxation then the relaxation of the basic graph pattern of a query.

3.2.1. Triple pattern relaxation

A triple pattern is composed of a subject, a predicate, and an object (s p o) $\in$ (I $\cup$ V) $\times$ (I $\cup$ V) $\times$ (I $\cup$ V $\cup$ L), where I is a set of IRIs (Internationalized URIs), L is a set of literals, and V is a set of variables disjoint from the sets I and L. Given two triple patterns tp and tp', tp' is a triple pattern logically derived from tp if one or more relaxation rules are applied to tp'. Relaxation rules are frequently based on the RDFS entailment rules (see Table 2).

- *Property relaxation* $\prec sp$ (based on rules 1-2) replaces a property *p* existing in the predicate of a triple pattern with its superproperty *p'*.
- *Type relaxation* $\prec sc$ (based on rules 3-4) replaces a class *c* existing in the subject or the object of a triple pattern with its superclass *c'*.
- *Simple relaxation* $\prec s$ replaces a constant (IRI or literal) by a variable. This relaxation is frequently used when it is not possible to apply property or type relaxations.
- *Typing relaxation using the domain* $\prec d$ *and the range* $\prec r$ (based on rules 5-6) replaces a triple pattern (?x, a, y) with (?x, type, c) or a triple pattern (x, a, ?y) with (?y, type, d) where the triples (a, domain, c) and (a, range, d) exist in the ontology.

Typing relaxations using the domain and range can have several disadvantages. Adding types is not helpful if they already exist in the query. It is also not possible to use domain or range relaxations when there is a variable in the object or the subject because information about the link of these variables and other terms in the query will be lost (i.e., joins will be broken). In addition, applying these relaxations makes it hard to compare the relaxed triple pattern with the original one. Hence, measuring their similarity would need uncommon methods. For these reasons, typing relaxation is not frequently used and so it is not considered in the rest of this section.

The set of relaxed triple patterns can be represented by an acyclic relaxation graph. This graph can be a lattice-based partial order where the relation $\prec$ is a triple pattern relaxation. A relaxation step transforms tp into a relaxed triple pattern tp' by replacing any element $e \in tp$ by $e' : tp' = tp \setminus e \cup e'$ where $e' \in \{\prec sp, \prec sc, \prec s\}$.

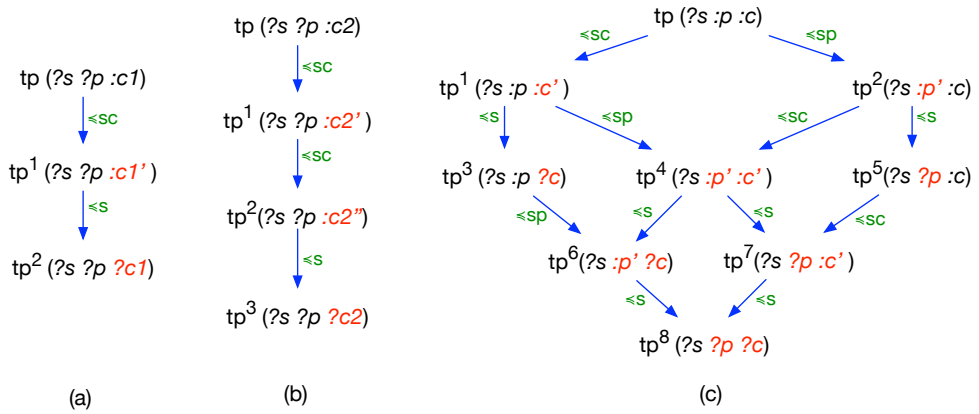


Fig. 2. Relaxation lattices of triple pattern *tp*. $\prec$sc is a class relaxation, $\prec$sp is a property relaxation and $\prec$s is a simple relaxation.

Figure 2 shows three relaxation lattices of a triple pattern tp. The size of a relaxation lattice depends on the relaxation possibilities allowed by the hierarchy of the ontology and the number of elements that is possible to relax. In Figure 2(a), one element of tp can be relaxed once (the object) while in Figure 2(b), it can be relaxed twice. In Figure 2(c), two terms can be relaxed once (the predicate and the object). The top-left corner of Figure 3, shows the class and property hierarchy used in these relaxation lattices.

The total size of the relaxation lattice is the cartesian product of the size of each element. The size of each element can be calculated as follows, where e is an element of a triple pattern:

$$size(e) = \begin{cases} 1 & \text{if } e \text{ is a variable} \\ 2 & \text{if } e \text{ is a constant} \\ 2+n & \text{where n is the number} \\ & \text{of superclasses of e} \end{cases}$$

3.2.2. *Query relaxation*

Let Q = X $\leftarrow$ P be a (conjuctive) query, where X = (x_1, ..., x_n) is a tuple of variables and P is a graph pattern composed of triple patterns. X is called the head and P is the body. Given two queries Q and Q', Q' is a query logically derived from Q, denoted $Q \prec_Q Q'$, if one or more relaxation rules are applied to Q. The set of relaxed queries can be expressed as a lattice-based partial order where the relation $\prec_Q$ is a query relaxation. This lattice is the product of the relaxation lattices of the triple patterns existing in the graph pattern. A relaxation step, noted Q $\prec_Q$ Q', transforms Q into a relaxed query Q' by replacing any element $e \in P$ by $e' : Q' = X \leftarrow (P \setminus e \cup e')$ where $e' \in \{\prec sp, \prec sc, \prec s\}$. In this lattice structure, the original query is the lower bound and the query whose triple patterns contain only variables is the upper bound. Let $Q = tp_1^0 \wedge \cdots \wedge tp_n^0$ be the original failing query with n triple patterns and Q' be the set of relaxed queries $\{Q' = tp_1^{i_1} \wedge \cdots \wedge tp_n^{i_m} \mid \exists c \in [1,m] : i_c > 0\}$ where c counts the number of relaxations of each tp.

Figure 3 shows an example of a relaxation lattice of a query Q. The graph pattern of Q is composed of the triple patterns of Figures 2(a) and 2(b). The relaxation distance between two queries is the number of relaxation steps applied to the most specific query to obtain the most general relaxed query. This lattice has five levels. Relaxed queries having the same distance from the original query are part of the same level. The size of a relaxation lattice is the product of the sizes of the relaxation lattices of its triple patterns. In this example, the lattice size is twelve. If the graph pattern is composed of several triple patterns and the relaxation possibilities of each triple pattern are important, then the size of the set of queries Q' can be huge. One of the main challenges facing query relaxation approaches is *how to choose the relaxed queries to be executed until efficiently obtaining k-relevant results*.

Queries can be distinguished based on the shape of their basic graph patterns into four types: star, chain, composite, and property path. In a star query, all triple patterns share the same variable in the subject or the object. If

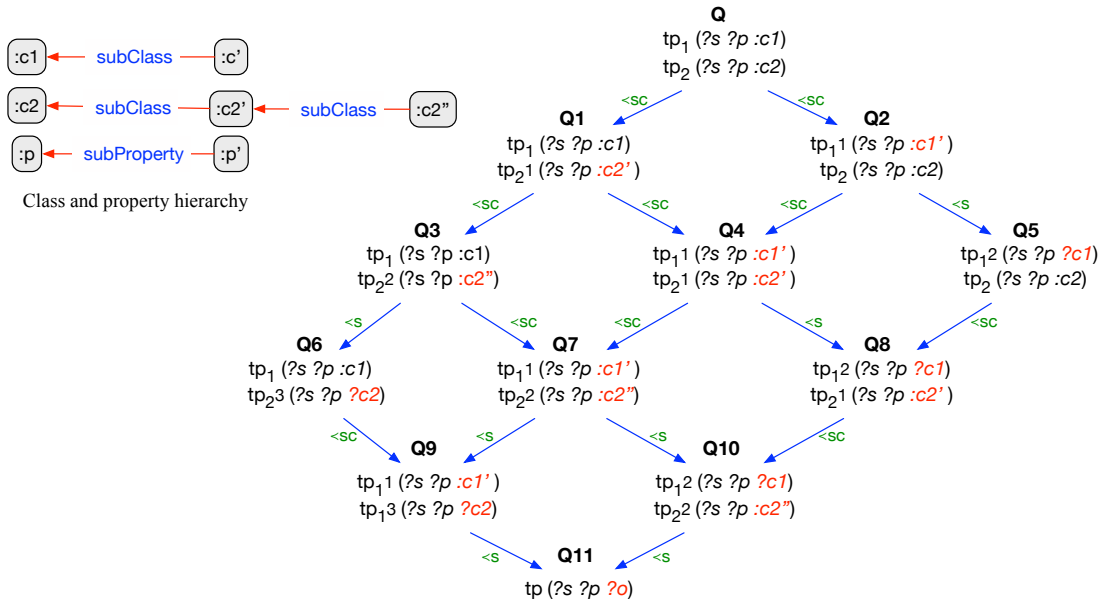


Fig. 3. Relaxation lattice of a query Q.

all triple patterns are centered around the same subject, it is called a subject star-shaped query. In a chain query, the object of a triple pattern is also the subject of another triple pattern. So the same variable exists in a subject and an object. A composite query is a combination of star and chain queries. Finally, a property path query is a query whose predicate is a regular expression over a set of properties and not only one property.

3.3. Information content measures

Analysing all relaxed queries existing in the relaxation lattice of a query is time-consuming and unnecessary. Information content [18], is frequently used to compute the similarity of relaxed queries to the original query. That is, to avoid the execution of an important number of relaxed queries until obtaining *k-relevant* results, several approaches use statistical information about the concerned dataset, such as the number of entities per class and the number of triples per property. The goal is to generate and execute relaxed queries from the most to the least similar to the original query.

Suppose $\mathcal{C}$ is a set of concepts in an is-a taxonomy. The similarity between two concepts is how much they share information in common. Then, according to the standard argumentation of information theory [19], the information content of concept C can be quantified as the *negative log-likelihood -log Pr(C)*. Where Pr(C) is the probability of finding an instance of concept C in a dataset which is computed as $Pr(C) = \frac{|Instances(C)|}{|Instances|}$. This implies that Pr(C) is monotonic over the taxonomy: if C_1 is-a C_2, then $\Pr(C_1) \leqslant \Pr(C_2)$.

Based on information content, [4, 8] propose similarity measures between terms, triple patterns and queries. These measures are used by several works because they allow ranking relaxed queries from the most to the least similar in a total order of relaxed queries.

3.3.1. Similarity between terms

The similarity between terms depends on their nature: class, property, constant, or variable. It is computed based on the instances of classes and properties in a dataset.

- *Similarity between classes* is $Sim(C, C') = \frac{IC(C')}{IC(C)}$ where $IC(C)$ is the information content of C:$-log\ Pr(C)$, where $Pr(C) = \frac{|Instances(C)|}{|Instances|}$ is the probability of finding an instance of class C in the RDF dataset.
  For example, if the subject or object of a triple pattern is a class c_1 and is relaxed to its superclass c_2 using

type relaxation, the similarity between c_1 and c_2 is $Sim(c_1, c_2)$.
Notice that, the similarity between classes is zero when all the instances in the RDF dataset belong to the superclass C', i.e., $Pr(C') = 1$ and thus $-log\ Pr(C') = 0$. Notice also that the similarity between classes is undefined when all the instances belong to the superclass C', and none to C, i.e., $Pr(C) = 0$ and thus $-log\ Pr(C') = undefined$.

- *Similarity between properties* is $Sim(P, P') = \frac{IC(P')}{IC(P)}$ where $IC(P)$ is the information content of P: $-log\ Pr(P)$, where $Pr(P) = \frac{|Triples(P)|}{|Triples|}$ is the probability of finding a property of P in triples of the RDF dataset.
  For example, if the predicate of a triple pattern is a property p_1 and is relaxed to its superproperty p_2 using property relaxation, the similarity between p_1 and p_2 is $Sim(p_1, p_2)$.
  Notice that same as the similarity between classes, the similarity between properties is zero when all the triples in the RDF dataset belong to the superproperty P', and the similarity between properties is undefined when all the triples belong to the superproperty P', and none to P.
- *Similarity between constants and variables* is $Sim(T_{const}, T_{var}) = 0$.
  For example, if the object of a triple pattern t_{const} is a class and is relaxed to a variable t_{var} using simple relaxation, the similarity between t_{const} and t_{var} is 0.

3.3.2. Similarity between triple patterns

Given two triple patterns tp and tp', such that $tp \prec tp'$, the similarity of the triple pattern tp' to the original triple pattern tp, denoted $Sim(tp, tp')$, is the average of the similarities between the terms of the triple patterns:

$$Sim(tp, tp') = \tfrac{1}{3} \,.\, Sim(s, s') + \tfrac{1}{3} \,.\, Sim(p, p') + \tfrac{1}{3} \,.\, Sim(o, o').$$

Where s, p, o, s', p' and o' are respectively the subject, predicate and object of the triple pattern tp and the relaxed triple pattern tp'. If tp' and tp'' are two relaxations obtained from tp and $tp' \prec tp''$ then $Sim(tp, tp') \geqslant Sim(tp, tp'')$.

3.3.3. Similarity between queries

Given two queries Q and Q', such that $Q \prec Q'$, the similarity of the original query Q' to the original query Q, denoted $Sim(Q, Q')$, is the product of the similarity between triple patterns of the query:

$$Sim(Q, Q') = \prod_{i=1}^{n} w_i . Sim(tp_i, tp'_i).$$

Where tp_i is a triple pattern of Q, tp'_i a triple pattern of Q', $tp_i \prec tp'_i$, and $w_i \in [0, 1]$ is the weight of triple patterns tp_i. Weight can be specified by the user to take into account the importance of a triple pattern tp_i in query Q. Thus $Sim(Q, Q') \in [0, 1]$ is a function that defines a total order among relaxed queries.

This similarity function is monotone, i.e., given two relaxed queries $Q'(tp'_1, ..., tp'_n)$ and $Q''(tp''_1, ..., tp''_n)$ of the user query Q, if $Q' \prec Q''$ then $Sim(Q, Q') \geqslant Sim(Q, Q'')$.

3.4. Relaxation based on the similarity of instances

Query relaxation based on the RDFS entailment can have effective results in queries over datasets where the hierarchies of classes or properties are rich. However, in some use cases, precise queries are posed for matching structured descriptions of resources in the RDF graphs. Relaxing these queries by replacing constants (that are actually resource instances) appearing in subjects or objects with *similar resources* is more appropriate than replacing them with variables (using simple relaxation). To measure the similarity among resources, similarity functions are needed to estimate the distance between resources. Different functions and methods can be used for computing the similarity between resources based on their type (numeric, string, date, etc.). For example, for dates, normalized distances can be used. For numeric values, Euclidian distances can be employed. String functions can be based on natural language techniques where lexical analysis is applied, like Jaccard similarity, Levenshtein distance, TF-IDF score, cosine similarity, etc.

For instance, supposing x_i and y_i are two numeric values, and min_i and max_i are the minimun and maximum values of a certain resource respectively, then the numeric distance between these two values could be the ratio of the difference between the values x_i and y_i over the difference between min_i and max_i.

A similarity measure is proposed in [20] known as semantic graph edit distance. This distance measures the similarity between RDF graphs by considering semantic meanings. Given two graphs, it computes the minimum cost required to transform a graph to another based on the semantic similarity between instances.

Reference	Contribution		Description
[2, 3, 5]	Extends the OPTIONAL clause with a RELAX clause.		Extends the OPTIONAL clause using the RDFS entailment rules and the *extended reduction of the ontology*.
[6]	Ranking model based on Bayesian networks.		Proposes a ranking model that orders the relaxed queries based on the similarity of their selectivity using Bayesian networks. Bayesian networks are build from the properties describing entities.
[4, 8]	Relaxation strategies based on information content.	BFSR	Selects gradually the best current relaxed query according to the similarity measure based on information content.
		OBFSR	Eliminates the unnecessary relaxed queries which do not contribute with new answers by studying the distribution of instances over the classes and predicates.
		BR	Identifies the necessary relaxed queries to obtain *top k* answers based on their selectivity.
[10]	Query relaxation strategies based on failure causes.	MBS	Prunes the query relaxation graph using the Minimal Failing Subqueries (MFS) of the original query.
		O-MBS	Optimizes MBS by focusing on the unrepaired MFS.
		F-MBS	Extends O-MBS by discovering all the MFS of every relaxed query.
[12]	Relaxing failing queries based on the hitting set problem (CADER).		Proposes a strategy to determine the maximal set of succeding subqueries of a query by applying the hitting set problem.
[13]	License-aware query relaxation (FLiQue)		Proposes a license-aware query relaxation strategy that supports federated queries and prevents license conflicts.
[7]	Statistical language model relaxation.		Uses NLP techniques (statistical language models) and the Jensen-Shannon divergence to relax entities and predicates.
[9]	Relaxations of heterogeneous resources descriptions.		Proposes a framework of multiple distance functions to map every pair of entity values into a relaxation score. Euclidian distance is used to define the distance between the entities of the original query and all the entities of the dataset.
[11]	Clustering-based relaxation.		Proposes optimized algorithms based on Formal Concept Analysis to define similarity clusters to find approximate answers. Its originality is the answers' partionning that begins from the most general query to most specific queries close to the original query.

Table 3

Table of query relaxation strategies and their descriptions.

4. Review of query relaxation approaches

We organise the survey of existing works in two sections. First, Section 4.1 analyses approaches focusing on relaxation using RDFS ontologies and entailment rules [2–6, 8, 10, 12, 13]. Then, Section 4.2 surveys works oriented to relaxation based on similarity of instances [7, 9, 11]. Table 3 shows the list of all the analysed relaxation approaches with a brief description of their main contributions. For each work we describe its approach, the experimental setup and evaluation conclusions.

Rule	Condition	Statement
(7)	(b, domain, c) AND (a, subProperty, b)	⇒ (a, domain, c)
(8)	(b, range, c) AND (a, subProperty, b)	⇒ (a, range, c)
(9)	(a, domain, b) AND (b, subClass, c)	⇒ (a, domain, c)
(10)	(a, range, b) AND (b, subClass, c)	⇒ (a, range, c)

Table 4
Domain and range rules of RDFS for the extended reduction of an RDFS ontology.

4.1. Relaxation based on ontologies and RDFS entailment rules

Most relaxation strategies use type relaxation, property relaxation, and simple relaxation. In general, the closure of the RDF dataset is considered. To improve the performance, dataset statistics are used to calculate similarity measures between a relaxed query and the original query.

4.1.1. Extension of the OPTIONAL clause with a RELAX clause

Hurtado et al. [2, 3, 5] propose a RELAX clause as a generalization of the OPTIONAL clause of SPARQL. The OPTIONAL clause allows a query to give results that do not match all constraints. More specifically, query results are obtained even if they do not match the triple patterns defined in the OPTIONAL clause.

These works use four types of relaxation: type relaxation, property relaxation, typing relaxation (using the domain and range rules), and simple relaxation. For performance reasons, they use the *extended reduction of the ontology* instead of the RDF closure obtained by applying entailement rules. The idea is to discard derivable triples and thus indirect relaxations. Table 4 shows range and domain rules of RDFS used for the extended reduction of an RDFS ontology.

Supposing a is a subproperty of b, then:

- Domain rules state that (rule 7) if b has a domain c, then it is entailed that a has domain c; (rule 8) if b has a range c, then it is entailed that a has range c.

Supposing a is a property and b is a subclass of c, then:

- Range rules state that (rule 9) if a has a domain b, then a will also have domain c; (rule 10) states that if a has a range b, then a will also have range c.

These approaches compute the extended reduction of the ontology as follows. Given an ontology and its closure (obtained by the entailment of rules of Table 2), the additional rules of Table 4 are applied in reverse order until no more rules can be applied. Then, rules 1 and 3 of Table 2 are applied in reverse order until no more rules can be applied. It is possible to apply a rule in reverse order by removing a triple that can be derived by the rule from the ontology. The extended reduction of the ontology thus allows only direct relaxations to be applied to queries that correspond to the smallest relaxation distance.

These works perform the breadth-first traversal of the relaxation lattice of a query and return the answers in a ranking order according to the relaxation distance. The notion of ranking is based on the structure of the relaxation lattice where answers are returned from the less to the more general from a logical standpoint. But the queries that are on the same relaxation level are not distinguished. For performance reasons, the redundancy of answers is avoided by taking into consideration logical subsumption, i.e., when the answers of a relaxed triple pattern t' are computed, only matchings that are not matchings of other relaxed triple patterns of t' are returned.

[2] studied the relaxation of queries over RDF data by considering queries with one single triple pattern. Later, [3] studied the same relaxation approach but addressed graph patterns. Then [5] extended the application of ontology-based relaxation from graph patterns to property path queries. To do this, authors convert a property path query into a chain query. Once a chain query is obtained, the query relaxation is done as usual.

In these works, all generated relaxed queries are executed, even those expected to give an empty result. This is a drawback in terms of execution time. In addition, two relaxed queries with the same relaxation distance in the relaxation lattice are not distinguished. The RELAX clause lacks of generalization to be applied to complex SPARQL queries, including disjunctions and optionals.

The authors in [2, 3, 5] do not evaluate their contributions experimentally.

4.1.2. Ranking model based on Bayesian networks

Huang et al. [6] propose a model to rank relaxed queries according to their similarity to the user query using Bayesian networks. The similarity between a relaxed query and the user query is measured in terms of the difference of their estimated results, i.e., their selectivity. This work relaxes only predicates and objects. Authors target subject star-shaped queries. The originality in estimating selectivity is to consider the correlation among properties describing entities. Thus, this work focuses on star queries searching for triples describing similar entities.

The selectivity of a query depends on the joint probability distribution over the values of the properties whose entities match the query. But the possible combinations of values of properties would be exponential. Therefore, this work proposes to use Bayesian networks as an appropriate structure to divide the joint probability distribution into several distributions of fewer variables by using the conditional independence assumption.

Properties describing entities in the dataset are grouped into *cluster-property tables*. Bayesian networks are constructed for each cluster-property table. The K2 algorithm [21] is used to learn the structure of the Bayesian networks. Learning the structure of Bayesian networks requires searching through the space of possible network structures and learning the probability parameters of every structure. This process aims to find the structure that best fits the observed data. Since the number of possible structures grows exponentially as a function of the number of variables, this process may be costly and it is done in an offline process previous to the online query processing.

Experimental setup. Experiments were conducted using Jena[11] and the Lehigh University Benchmark (LUBM) [22]. LUBM is based on a university domain ontology that includes 32 properties and 43 classes. The dataset used in the experiments contained 600K triples. five star queries were developed for the experiments, where each query is composed of three to four triple patterns. The implemented algorithm to compute the *k-relevant* approximate answers is based on best-first search. The parameter k was fixed to 50 and 150.

Evaluation. Experimental results show that this relaxation strategy can avoid some unneccessary relaxations by considering the real data distribution, i.e., relaxations giving empty results are avoided. However, the experiments carried out do not present a comparison with other state-of-the-art contributions. Thus, there is no confirmation of the better performance brought by this work compared to other contributions.

4.1.3. Relaxation strategies based on information content

Huang et al. [4, 8] propose strategies that ensure the quality of the query results by computing the similarities of the relaxed queries using the information content. [8] proposes and evaluates the performance of three relaxation strategies: Best First Strategy (BFSR), Optimized Best First Strategy (OBFSR) and Batch-based Relaxation Strategy (BR). These strategies are based on the best-first traversal strategy of the relaxation graph and execute the relaxed queries in a ranking order according to similarity measures to generate the *k-relevant* answers for a query. Similarity between queries is calculated as explained in Section 3.3.

- BFSR selects the best-relaxed query from the candidates queries at every level of the query relaxation lattice until enough answers are returned. This query is chosen based on the similarity measures.
- OBFSR considers the join dependency between the triple patterns and the data distribution in the dataset. A relaxed triple pattern may have no new answers in the dataset. This approach studies the two cases where the relaxed queries do not contribute to new answers: (1) generalizing the entities (subject or object), and (2) generalizing the properties. At the relaxation stage, if a class is replaced with its superclass in a triple pattern of a query, OBFSR checks if a superclass has more instances than its subclass. The relaxed query can generate new answers if a superclass has more instances. The same process is applied when a property is replaced by its superproperty. Therefore, OBFSR checks the usefulness of relaxed subqueries and skips unnecessary relaxed queries.

[11] https://jena.apache.org/

– BR studies the problem of relaxing queries as a batch-based process. It improves OBFSR by using the answers' subsumption hypothesis between relaxed queries. Queries and their relaxed queries are set in a batch. The relaxed queries that are subsumed by others in a batch are skiped by the BR algorithm to reduce the overall execution time at avoiding redundancy of answers calculation. The critical problem of this approach is estimating the size of the batch, i.e., the number of relaxed queries needed to obtain *k-relevant* answers. So, the primary process is divided into three steps: predicting the batch size, executing the necessary queries, and ranking the answers. This approach estimates the selectivity of queries to predict the batch size. The selectivity of a conjunctive query is defined as the product of the selectivity of triple patterns. The selectivity of a triple pattern depends on the selectivity of every element computed using the dataset statistics. Using these definitions, the selectivity of a set of relaxed queries is estimated considering the overlappings among the relaxed queries. Hence, the estimation of the number of answers returned by a set of relaxed queries is the estimation of the selectivity of this set multiplied by the number of triples in the dataset.

Experimental setup. These algorithms were evaluated using Jena TDB[12]. [8] constructed a dataset using LUBM. The generated dataset contained about 10M distinct triples. In addition, seven queries were designed to conduct experiments that evaluate the performance of BFSR, OBFSR and BR algorithms with the increase of k (number of approximate answers). Three types of queries are used in these experiments: star queries, chain queries, and composite queries. The number of triple patterns in every query ranges from two to five. The parameter k was set at 10 to study the execution time and the number of executed relaxed queries for each query then increased to 50 and 150.

Evaluation. With the increase of k, more relaxed queries need to be executed. BR skips relaxed queries that are subsumed by others and reduces the overall execution time. Therefore, BR is suggested when the user demands more approximate answers. Otherwise, when the user asks for few answers, BR will behave as OBFSR and BFSR. However, no difference in results quality is observed according to the type of queries used in the experiments. This means that the type of queries do not have an impact on the quality of results obtained by these strategies.

4.1.4. Query relaxation strategies based on failure causes

Foukou et al. [10] propose strategies to prune the query relaxation lattice by identifying the minimal sets of (conjunctive) triple patterns in failing queries that fail to return answers. These failing sets of triple patterns are called Minimal Failing Subqueries (MFS). Authors argue that executing relaxed queries in a similarity-based ranking order lacks of efficiency because queries are relaxed without identifying their failure causes. Identifying the set of MFS of a query is an NP-hard problem. In [12, 23], authors propose efficient methods for the computation of a set of MFS of a failing SPARQL query. Discovering a set of MFSs of a query Q roughly consists in (1) finding an MFS of Q, (2) computing the maximal queries not containing the MFS determined in the first step, and (3) repeating the first two steps until no more MFS are found.

[10] proposes three different strategies based on failing subqueries, MFS-Based Search (MBS), Optimized MFS-Based Search (O-MBS), and Full MFS-Based Search (F-MBS). These strategies calculate query similarities as introduced in Section 3.3. The three proposed strategies use different levels of information about failing queries.

– MBS explores every query in a queue of relaxed queries totally ordered according to their similarity measures. The failure causes of the original query Q are identified, i.e., the set of MFS. If a relaxed query contains an MFS of Q, then it is considered as failing. MBS prunes the query relaxation graph with the identified failing queries. This strategy improves the query relaxation efficiency but its limitation is that it does not discover all failing subqueries because it focuses only on the failing causes of the initial query.
– O-MBS optimizes MBS by focusing on the MFSs that are not *repaired*. MFS existing in a query must be relaxed (i.e., repaired), otherwise, the query fails in producing results. Relaxed queries where the MFS are not relaxed are considered unnecessary. Intuitively, a relaxed query Q' of Q fails if and only if at least one MFS has not been repaired in Q'. Repaired queries are executed and if they give no result they are pruned as well as all those relaxed queries having the same repaired MFS. This approach is better than MBS because it checks

[12] https://jena.apache.org/documentation/tdb/

failing relaxed queries that are not discovered by MBS. Nevertheless, all failing causes of a relaxed query are not necesarily discovered.
- F-MBS extends the O-MBS strategy and discovers all the MFS of every relaxed query Q'. Although this extended strategy discovers all possible failing subqueries, its drawback is that it can take more execution time compared to the O-MBS strategy.

Experimental setup. These three algorithms were implemented in Java using Jena TDB and Virtuoso[13]. Datasets generated with LUBM that range from LUBM100 (17M triples) to LUBM1K (167M triples) were generated. Seven queries are used in the evaluation experiments, where these queries include the main query patterns (star, chain, and composite). These query patterns range between 1 and 15 triple patterns and include 1 up to 4 MFSs. k is set to 50.

Evaluation. According to the experiments, O-MBS is the best strategy since it minimizes the computation time and the number of executed queries in most cases. O-MBS is a good compromise between MBS and F-MBS.

4.1.5. Relaxing failing queries based on the hitting set problem

Mebrek et al. [12] propose an approach called *CADER* for computing relaxations for failing queries. This approach proposes to eliminate triple patterns from the query. It does not use ontology-based relaxation. It involves two main steps: (1) determining all the minimal failing subqueries (MFSes), and (2) exploring these failing subqueries to compute the set of maximal succeeding subqueries (XSSes).

The basic idea of the first step is to decompose the initial query into subqueries and browse them, starting from those having the lowest number of triple patterns to the biggest. This iterative browsing reduces the exponential search space. If a query fails to give answers, then all its proper superqueries will fail. Hence, it is unnecessary to evaluate the following superqueries.

Regarding the second step, this approach models the query relaxation problem as a hitting set problem. Given a collection of subsets, the hitting set problem is finding the smallest subset that intersects (hits) every set in the collection. It aims to compute the set of relaxed queries, i.e., XSSes. Once the MFSes are computed as discussed above, the set of relaxed queries can then be computed without any access to the RDF dataset. This is performed by discovering the set of MFSes and searching for their hitting sets which correspond to the complement set of XSSes. Hence, the set of succeeding queries are the set of maximal subqueries that succeed to give answers, and these queries are just generated by eliminating some triple patterns from the original query. These queries are the relaxed queries generated at the end of this strategy.

Experimental setup. CADER has been implemented in Java with the Jena library. The experiments are carried out using datasets from DBpedia and LUBM. Using the LUBM data generator, tests sets were created ranging from 65M to 2 billion triples for obtaining datasets of different sizes. 21 queries of varying types (star, chain, and composite queries) were generated over the DBpedia dataset and 197 queries for the LUBM dataset having between 2 and 15 triple patterns. The LUBM tested queries were borrowed from [23].

Evaluation. Several experiments were performed to compare the proposed approach with the LBA and MBA algorithms proposed in [23]. Results show that *CADER* behaves the best regardless the types of queries in terms of execution time.

4.1.6. License-aware query relaxation

Moreau et al. [13] proposes FLiQue, a license-aware query processing strategy that uses query relaxation to deal with contradictions among licenses. When two or more licensed datasets participate in evaluating a federated query, to be reusable, the query result must be protected by a license compliant with each license of the involved datasets. This approach detects and prevents license conflicts during federated query processing. If a license conflict is detected, FLiQue dynamically discards datasets of conflicting licenses. As this solution may generate an empty query result, the original user query is relaxed with ontology-based relaxation techniques. Thus, given a SPARQL query and a federation of licensed datasets, the goal is to guarantee a relevant and non-empty query result whose license is compliant with each license of involved datasets. The challenge is to limit the communication cost when the relaxation process is necessary.

[13]https://virtuoso.openlinksw.com/

To reduce the overhead induced by the distributed query relaxation process, FLiQue uses data summaries, statistics, and descriptions of data. It uses a compatibility graph of licenses that can be produced with Cali [24], a lattice-based model for license orderings. Cali automatically positions a license over a set of licenses in terms of compatibility and compliance. The originality is to pass through a restrictiveness relation to partially order licenses. FLiQue searches for licenses that are compliant with each license of the datasets that can participate in the query evaluation. Subfederations that avoid license conflicts are defined if no compliant license exists in the original federation. If no subfederation can execute the query, then the O-MBS approach (cf. Section 4.1.4) is used to produce and propose relaxed queries. This work does not propose a new ranking strategy (queries are ordered by similarity like in O-MBS), but it is able to propose queries ordered by the restrictiveness of concerned licenses. The idea is that the more a license is restrictive, the less the content can be reused. So, queries whose evaluation datasets have less restrictive licenses should be preferred.

Experimental setup. FLiQue was implemented in Java using Jena and CostFed [25], an index-assisted federated engine for SPARQL endpoints which relies on a join-aware triple-wise source selection by considering URI prefixes. The conducted experiments for the evaluation of FLiQue use Virtuoso endpoints and LargeRDFBench [26], a benchmark with more than 1 billion triples for a federation of SPARQL endpoints. This benchmark contains 32 federated queries (14 simple queries, 10 complex queries, and 8 large queries). The number of triple patterns ranges from 2 to 12. Queries were executed over a federation of 11 real interlinked data sources (DBpedia, IMDB, Semantic Wew Dog Food, Linked MDB, Geonames, New York Times, Affymetrix, ChEBI, KEGG, Drug Bank, Linked TCGA, and Jamendo).

Evaluation. In the experiments, queries are executed until obtaining the first result (k=1) as the goal of this work is to show that it is possible to preserve licenses during federated query processing. For every query, experiments were carried out to compare the time to get the first result of the query with and without FLiQue. FLiQue shows a constant overhead due to checking the license conflicts. The experiments showed that on average FLiQue generated 21.4 failing relaxed queries and executed 1.75 failing relaxed queries with the benchmark used. The maximal number of relaxed queries was 69, but only 3 were executed until finding the first one with non-empty result. According to the experimental results, FLiQue succeeds in limiting the communication costs during the relaxation of federated queries whose results set cannot be licensed. However, LargeRDFBench is not ideal for this evaluation since the benchmark queries contain a lot of variables and very few classes. In addition, the potential for relaxation is poor due to the short deep of the hierarchies of the ontologies concerned by LargeRDFBench datasets.

4.2. Relaxations based on similarity of instances

Upcoming works propose relaxation approaches that concentrate on rewriting queries based on instance similarities. They focus on discovering entities similar to the ones specified in the user query. RDF graphs describe entities, thus, similarities can be identified according to numeric or symbolic distances among entities' descriptions.

4.2.1. Statistical language model relaxations

Elbassouni et al. [7] see the query relaxation problem as the entity-relationship-oriented counterpart of query expansion in the traditional keyword-search setting. It uses NLP techniques, particularly statistical language models, to relax queries. Statistical language modeling uses statistical and probabilistic techniques to predict the next word in a given sequence of words. This is performed by learning the probability of word occurence. In the same way, probability of occuring two entities together in the same triple in the dataset, is learned. Then similar entities are detected based on this statistics. This work proposes to rewrite triple patterns using similar entities and similar relations. Similarities are calculated using the entity descriptions existing in the knowledge graph. Each entity generates a document containing the triples where the entity appears as subject or object. Each relation generates also a document containing the triples where the relation appears as predicate. Then unigrams and bigrams are defined from each document. Unigrams correspond to all entities, and bigrams correspond to all entity-relation pairs. Language models are then estimated by mixing unigrams and bigrams with linear interpolation smoothing from the corpus (i.e., the knowledge graph). The language model of every entity or relation is thus a probability distribution. The distance between the language models of every pair of entities allows to identify similar entities. The similarity score between any two distributions is computed using Jensen-Shannon divergence. Thus, for each

entity, this approach computes a ranked list of similar entities. This processing must be done offline previous to the online query processing.

Three types of relaxations are proposed: (1) replacing entities (subjects or objects) and properties with other related entities and properties, (2) replacing entities and properties with variables, and (3) removing triple patterns or making them optional.

Experimental setup. The effectiveness of this relaxation approach was evaluated in two experiments. The first one evaluated the quality of relaxations of entities and relations. The second one evaluated the quality of the final results obtained from the original and relaxed queries. The experiments were conducted over two datasets. The first dataset was derived from the LibraryThing community, and the second dataset was derived from a subset of the Internet Movies Database (IMDB). The data from both sources was automatically parsed and converted into RDF triples. The number of triples was over 700,000 and 600,000 for LibraryThing and IMDB, respectively. In addition, 40 queries were constructed for each dataset.

Evaluation. The top 5 relaxations were obtained for every entity in the experiment concerning the quality of relaxations. These observations were presented to 6 evaluators using the Amazon Mechanical Turk service[14]. Evaluators studied the quality of these relaxations and how close they were. The same was carried out for the relations. At this stage, the evaluators agreed with the ranking results of the entity and relations relaxations indicating the good performance of this contribution in relaxing entities and relations. A further experiment was also performed to study the quality of the relaxed query results. In this experiment, the top five relaxed queries were studied to evaluate the quality of relaxed queries overall. The ranking of the query results in this experiment significantly meets the evaluators' expectations. It is important to emphasize that this work considers both the quality of the results from the user's point of view and the closeness of the relaxed query to the original query.

4.2.2. Relaxations of heterogenous resources descriptions

Hogan et al. [9] focus on entity-lookup queries using similarities of RDF terms. They address *vague queries* executed over heterogenous and incomplete data. Vague here means that the query issuer may be not sure about the query constraints. If the query has no results, it can be rewritten with similar constraints. Thus, this work proposes a generic framework to extract distance scores between resources based on their structured descriptions. This processing must be done offline previous to the online query processing. The target queries have variables on the subject and only objects are relaxed.

The proposed framework proposes to use multiple distance functions (called matchers) to map a pair of entity values into a relaxation score: a value in [0,1] where 1 indicates that the two values are not interchangeable and 0 indicates perfect interchangeability. For instance, M(:blue, :navy) might be 0.2, suggesting :navy as a good relaxation for generic :blue. Distance functions depend on the attributes' types. For instance, normalized distances can be used for numeric attributes like maximum, mean, minimum, etc. A function based on the Levenshtein distance can be used for string attributes. Also, background knowledge can be used like established similarity tables (for example, for colors). When a sufficient corpus exists, these similarity tables can be generated with instance-matching techniques like those proposed by Hogan et al. in [27].

The proposed approach considers the original query as the origin, the matchers map entities onto points in an n-dimensional euclidean space with each dimension ranging over [0,1]. When an entity has multiple values for a given attribute, the closest to the query is used. The overal distance from the original query to each entity gives an overall relaxation score that can be used to rank results. Further, users can annotate the attribute-value pairs of the original query with a *vagueness score* to control the relaxation by property. [9] also introduced another way to compute the similarity between attributes, named *concurrence*, which is based on the cardinality of property-value pairs in the database.

Experimental setup. The proposed relaxation framework was tested against the vehicles dataset of the EADS project (European Aeronautic Defence and Space Company). This dataset contained 50K RDF triples. The effectiveness of this framework was evaluated over three queries having two or three triple patterns. These queries were defined from their expression in natural language. Then, handmade, the expressions were transformed into SPARQL

[14]https://www.mturk.com/

queries with vagueness scores. Transforming natural language phrases automatically into queries is out of the scope of this paper.

Evaluation. In the experimental evaluation, relaxations were carried out on the three defined queries with vaguness scores. The top 5 approximate answers of each query were collected and evaluated using scores based on root-mean-square deviation. From these observations, it is deduced that this approach has a weakness due to the functions declared in the framework. This weakness states that different functions may produce very similar values. Hence, it will be hard to compare these values and choose the best relaxation.

4.2.3. Clustering-based relaxation

Ferré [11] proposes a strategy for finding approximate answers and optimizing the computation of matchings of a relaxed query. This work allows query relaxation to be applied effectively on large queries and not only on queries with few triple patterns. The usage of ontology in relaxing queries is not mandatory. Thus, this strategy is considered an instance-based relaxation. Furthermore, this contribution considers query relaxation as moving from an RDF node n to semantically similar nodes. This contribution is based on formal concept analysis [28], which is a mathematical theory of data analysis for deriving implicit relationships between objects described through a set of attributes. Hence, the body of the query is considered as a description of a node, and the relaxed queries describe the semantic similarity of the similar nodes, which is symbolic. In addition, the set of similar nodes is the set of proper answers of a relaxed query, defined as the approximate answers of this query, which are not found in more specific relaxed queries. This paper proposes two algorithms: Answers Partitioning and Lazy Join.

- Answers Partitioning. In this strategy, an initial cluster represents the most general query. This general cluster is split into more fine-grained groups by adding triple patterns, i.e., more constraints over the graph. The choice of triple patterns is free from the set of triple patterns in the original query. Adding triple patterns progressively eliminates answers. The answers of a cluster are divided into two groups: the set of answers that satisfy the constraint and its complement with respect to the initial cluster. Passing from a cluster to its child cluster refers to passing from a general query to a more specific query in the lattice. This technique can be seen as a bottom-up approach instead of a top-down as usually employed to navigate over the query relaxation lattice. The advantage of moving from a more general query to a more specific one is that the evaluation of the new relaxed queries will not be exhaustive. That is because the new cluster will be the intersection of the instances of the previous cluster with the instances that satisfy the added triple pattern. This skips the unnecessary steps that involve computing again redundant answers.
- Lazy Join. The Lazy join algorithm is proposed to optimize the evaluation of the relaxed queries generated by the Answers Partitioning algorithm. When passing from a query to one of its possible relaxations in the lattice, new computation of matchings of the relaxed queries is necessary to obtain all the answers of the generated relaxed query. This computation is not trivial and can be costly due to the global joining of all the triple patterns. However, passing reversely from the relaxed queries to the more specific queries guarantees less complexity since the newly generated answers are a subset of the previous one. Furthermore, this means that the computation will involve only one small local join between the instances of the last query (more general query matchings) and the newly selected triple pattern (more specific query matchings). Hence, specific answers are determined gradually by considering local joins at every level where a new triple pattern is added. An advantage of this, is also that queries with multi-valued properties are efficient. If a multi-valued property exists in a query, this query may consists in several triple patterns joined through this property but with different object values. In general, this multi-valued property can lead to an explosion of the size of the joins of the related triple patterns. But using the Lazy join algorithm, this global join will be replaced with several small local joins.

In this work, entities (or constants) appearing in triple patterns (in subjects or objects) are replaced by filters. So, every occurrence of a common entity is replaced by a new variable, and a filter is added to the query setting this variable to the specified entity. Then, a simple relaxation consists in removing a filter.

Experimental setup. This approach was implemented in OCaml[15], and experiments were conducted to evaluate its efficiency and effectiveness with this implementation.[16] Three different datasets were used: MONDIAL[17], LUBM10, and LUBM100. The MONDIAL dataset contains 10K nodes and 12K triples. LUBM10 contains 315K nodes and 1.3M triples, and LUBM100 is 10 times bigger (3M nodes, 13M triples). Both datasets are interesting in this work since they are both rich in multi-valued properties. LUBM was used in the evaluation experiments to study the performance of this work in terms of ontology. While the MONDIAL dataset had a good role in studying the efficiency of similarity search using the rich node descriptions of this dataset. For every dataset, sets of small and large queries were both considered. Small queries for MONDIAL, had between 5 and 21 triple patterns. For LUBM, the seven queries of [8] having between 3 and 7 triple patterns were reused. Very large queries were tested. Queries up to 248 triple patterns were defined for MONDIAL. And queries up to 1505 triple patterns for LUBM.

Evaluation. Two experiments were performed to evaluate this approach. In the first experiment, small queries were executed to study the efficiency of the proposed algorithms in different execution modes. The execution modes consist in setting a limit to the relaxation distance and to the computation time. In the second experiment, large queries were executed to study the efficiency of proposed algorithms in similarity search. The experimental results show that these algorithms are efficiently capable of handling the problem of applying many relaxation steps to large queries with hundreds of triple patterns. For example, it is possible to explore the relaxation space of queries having up to 1500 triple patterns in a matter of minutes. But still some strategies must be considered for choosing the proper cluster to split and the splitting element that allow to retrieve good results at earlier levels of relaxations.

5. Comparison of relaxation approaches

This section compares the reviewed contributions in three parts, each part is summarized in a comparative table. Tables are organized horizontally in two sets of works. The first set contains works introduced in Section 4.1 (relaxation based on ontologies and RDFS entailment). The second one contains works of Section 4.2 (relaxation based on the similarity of instances). Section 5 highlights general differences and similarities of analysed works. Section 5.2 focuses on several aspects of the relaxation strategies. Then Section 5.3 underlines the evaluation and comparation efforts of studied proposals. Finally, Section 5.4 analyses these approaches and sheds the lights on the main conclusions of our study.

In the comparative tables, ✓ means the criteria is satisfied, ✓ means that the criteria is more or less satisfied, and ✗ indicates absence of the comparative criteria or absence of information.

5.1. General comparision of analysed works

This section provides a general comparison of the analysed relaxation strategies. Table 5 summarizes this comparision.

Query shapes. As we can see in the second column of Table 5, all works relax star queries. Most of them relax also chain and composite queries. [2] is limited to the relaxation of single triple patterns. But the extension of this work in [5] also proposes how to deal with regular path queries. Only [13] focuses on federated queries. We remark that some of analysed works focus exclusively on subject star-shaped queries.

Necessary information. The third column of Table 5 shows the necessary information of analysed works. Concerning the first set of analysed contributions, either the approaches use the dataset instances or data summaries. Instances are used to execute relaxed queries, the goal is to progressively obtain k results. Data summaries are used to calculate the similarity of queries based on information content in [10, 13]. This technique allows

[15] https://bitbucket.org/sebferre/sewelis/src/master/

[16] Recently, authors published a java implementation for the Answers Partition and Lazy Join algorithms but no experiments are published https://gitlab.inria.fr/hayats/CONNOR

[17] http://www.informatik-uni-freiburg.de/~may/Mondial/

Contribution	Query Shape				Necessary Information			Relaxed Terms		
	Star Queries	Chain Queries	Composite Queries	Other	Instances	Ontology	Data Summaries	Subject	Predicate	Object
Extension of the OPTIONAL clause with a RELAX clause [2, 3, 5]	✓	✓	✓	Regular path queries	✓	✓	✗	✓	✓	✓
Ranking model based on Bayesian networks [6]	✓	✗	✗		✓	✓	✗	✗	✓	✓
BFSR [8]	✓	✓	✓		✓	✓	✗	✓	✓	✓
OBFSR [8]	✓	✓	✓		✓	✓	✗	✓	✓	✓
BR [8]	✓	✓	✓		✓	✓	✗	✓	✓	✓
MBS [10]	✓	✓	✓		✗	✓	✓	✓	✓	✓
O-MBS [10]	✓	✓	✓		✗	✓	✓	✓	✓	✓
F-MBS [10]	✓	✓	✓		✗	✓	✓	✓	✓	✓
CADER [12]	✓	✓	✓		✓	✗	✗	✗	✗	✗
FLiQue [13]	✓	✓	✓	Federated queries	✗	✓	✓	✓	✓	✓
Statistical language model relaxation [7]	✓	✓	✓		✓	✗	✗	✓	✓	✓
Relaxation of heterogeneous resources descriptions [9]	✓	✗	✗		✓	✗	✗	✗	✗	✓
Clustering-based relaxation [11]	✓	✓	✓		✓	✓	✗	✓	✓	✓

Table 5
General comparison of analysed works.

to avoid executing an important number of relaxed queries. Data summaries are also used to limit the communication overhead in a federated environment in [13]. Instances are needed for constructing the Bayesian networks in [6], which helps in estimating the similarity measure of every relaxed queries.
All approaches of the first set of works, except for CADER, need the dataset ontology. CADER analyses the original triple patterns of a user query to identify the maximal subsets of not failing triple patterns.
It is evident that all contributions of the second part of our analysis need dataset instances. As an option, [11] can use the dataset ontology. None of these works use data summaries.

Terms relaxed. The fourth column of Table 5 shows that subjects are relaxed by almost all analysed works. [6, 9] address only subject star-shaped queries, so the subject is always a variable that is never relaxed. Predicates are relaxed by almost all works. All analysed works relax objects of triple patterns. [9] relax only objects with their framework of mapping functions with similarity distances among object values. [12] does not relax queries ontologically, it only eliminates failing triple patterns.

5.2. Comparison of query relaxation approaches

Further extensive comparisons are summarized in Table 6 based on several aspects of relaxation strategies.

Ontology relaxation. Second column of Table 6 summarizes the type of the relaxation applied when ontology relaxation is employed. Ontology relaxation is a way of relaxing queries by generalizing the initial query exploiting the ontology hierarchie. Only the contribution proposed in [2, 3, 5] mention using typing relaxation using the range and domain. We recall that this kind of relaxation can be impractical to calculate the query similarity or when types already exist in the initial query. But, [2, 3, 5] do not use query similarity. Most of the relaxation approaches use type and property relaxations except few contributions that focus only on looking for similar instances or eliminating triple patterns [7, 9, 12].

Contribution	Ontology-based relaxation			Simple relaxation	Relaxation based on similarity of instances	Lattice pruning	Techniques to avoid redundancy	Techniques for query ranking
	Type Relaxation	Property Relaxation	Typing Relaxation					
Extension of the OPTIONAL clause with a RELAX clause [2, 3, 5]	✓	✓	✓	✓	✗	✗	✓	Relaxation distance
Ranking model based on Bayesian networks to efficiently relax star queries [6]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric*
BFSR [8]	✓	✓	✗	✓	✗	✗	✗	Relaxation distance + similarity metric*
OBFSR [8]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric*
BR [8]	✓	✓	✗	✓	✗	✓	✓	Relaxation distance + similarity metric*
MBS [10]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric*
O-MBS [10]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric*
F-MBS [10]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric*
CADER [12]	✗	✗	✗	✓	✗	✓	✗	✗
FLiQue [13]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric*
Statistical language model relaxation [7]	✗	✗	✗	✓	✓	✗	✗	Statistical language-model based ranking
Relaxations of heterogenous resources descriptions [9]	✗	✗	✗	✓	✓	✗	✗	Normalized Euclidean distance/edit distances/concurrence algorithm
Clustering-based relaxation [11]	✓	✓	✗	✓	✓	✗	✓	Relaxation distance and symbolic similarity

Table 6

Comparison of query relaxation approaches.

*: similarity metric based on information content explained in Section 3.3.

Simple relaxation. Third column on Table 6 indicates that all the analysed aproaches use simple relaxation. This relaxation replaces the value of a term (instance or constant) with a variable. This will lead to finally obtaining the most general query that is composed of only variables.

Relaxation based on similarity of instances. Fourth column of Table 6 higlights the second set of analysed works that focus on the similarity of intances. Queries are relaxed based on the similarity between the database

instances and not on the ontology hierarchy. The similarity of instances is frequently computed using NLP techniques like word embedding, clustering, or probabilistic models. Several distance functions are often used to compute the semantic similarity between different entity values. That is because the similarity of instances is data-type dependent.

Lattice Pruning. Fifth column of Table 6 shows that some contributions propose techniques to prune the query relaxation lattice that can be huge (cf. Section 3.2.2). This process is relevant because performance reasons. [8, 10] studied the reason behind the failure of the relaxed queries. MBS and O-OMBS can discover some relaxed queries that do not give answers without executing them (partial pruning). While F-MBS finds all the minimal failing subqueries of relaxed queries. FliQue uses O-MBS but also a join-aware triple-wise source selection to prune unnecessary relaxed queries. OBFSR and BR also prune the lattice by studying the cases where a superclass or a superproperty does not contribute with additional answers. But BR executes relaxed queries in a batch. With the increase of the number of relaxed queries to be executed, it is more likely that these queries are subsumed by others in a batch. Hence, BR could skip these queries and outperforms BFSR when users expect more approximate answers.

Techniques to avoid redundancy. Sixth column of Table 6 shows that few contributions propose techniques to avoid redundancy. When a query Q is relaxed to Q', the relaxed query Q' may have common answers with Q leading to redundancy during the query processing. This redundancy is caused by the inheritance of classes and properties. Fetching same answers from the database several times is time and memory consuming. Hence it is a good advantage for the relaxation approach to be optimized to avoid redundancy. Relaxation approaches proposed in [2, 3, 5, 11] and the BR algorithm [8] make efforts to reduce redundancy. [2, 3, 5] avoid redundancy by considering the logical subsumption between queries. Lazy join algorithm proposed in [11] also results in calculating non-redundant answers by optimizing joins. The BR algorithm [8] skips the execution of relaxed queries that result in redundant answers by estimating the selectivity of the queries.

Techniques for query ranking. When a user query produces not enough results, the query relaxation possibilites can be huge. To propose the most close results to the user expectations, techniques for query ranking are used by almost all analysed works. This is shown in the last column of Table 6. The query relaxation lattice is a partial order. The relaxation distance between two queries gives a hint about the semantic similarity of their results. But the relaxed queries at the same level of the lattice can not be compared. Hence additional information can be used to be able to compare all the relaxed queries. The similarity of queries is used as a measure that allows to obtain a total order of relaxed queries. This measure is based on information content (cf. Section 3.3). [2, 3, 5] rank queries according to their relaxation distance. Similarity of queries based on information content is used by [6, 8, 10, 13]. Other works employ different ways of computing similarity, such as word embeddings methods [29], symbolic similarity [11], numeric distance functions [9], JS-divergence between statistical models [7]. CADER [12] does not consider ranking queries. It just discovers the set of relaxed queries without any order.

5.3. Comparision of experimental evaluations

Table 7 shows the comparison of the relaxation approaches based on the experimental evaluations. [2, 3, 5] do not involve implementations and experiments for validation. These works analysed the theoretical part of query relaxation as they were the first to propose relaxing using RDFS entailment and RDFS ontologies.

Used dataset. Second column of Table 7 underlines a clear tendency to evaluate query relaxation with the LUBM benchmark. That is because LUBM proposes a well designed OWL ontology containing interesting class and property hierarchies which are necessary to experiment with ontology-based relaxation. CADER uses two datasets (LUBM and DBpedia) to compare the experimental results with LBA and MBA algorithms proposed in [23]. Besides using LUBM, [11] uses MONDIAL to study the performance of the proposed approach without relaxation and in presence of multi-valued properties. FLiQue uses LargeRDFBench that composes a federation of 11 real interlinked datasets. FLiQue highlights the fact that the ontologies of these datasets are not deep enough to allow query relaxation. At the time, LargeRDFBench was the only benchmark for

Contributions	Used dataset	Number of queries	Size of queries	Comparision with other works	Available source code
Extension of the OPTIONAL clause with a RELAX clause [2, 3, 5]	-	-	-	✗	✗
Ranking model based on Bayesian networks [6]	LUBM	5 queries	3 to 4 triple patterns	✗	✗
BFSR, OBFSR, BR [8]	LUBM	7 queries	2 to 5 triple patterns	✗	✗
MBS, O-MBS, F-MBS [10]	LUBM	7 queries	1 to 15 triple patterns	Compared with BFSR [8]	✓
CADER [12]	LUBM and DBpedia	21 queries for DBpedia, 197 queries for LUBM	2 to 15 triple patterns	Compared with LBA and MBA [23]	✗
FLiQue [13]	LargeRDFBench	32 federated queries	2 to 12 triple patterns	✗	✓
Statistical language model relaxation [7]	2 datasets generated from LibraryThing community and IMDB	40 queries for each dataset	1 to 4 triple patterns	✗	✗
Relaxations of heterogenous resources descriptions [9]	Vehicles dataset of the EADS project	3 queries	2 to 3 triple patterns	✗	✗
Clustering-based relaxation [11]	MONDIAL and LUBM	7 queries for each dataset	Small queries (up to 21 triple patterns), large queries (up to 1505 triple patterns)	✗	✓

Table 7
Comparision of experimental evaluations.

federated queries. [7] tested their approaches on parts of the LibraryThing and IMDB datasets. These datasets were automatically parsed and converted into RDF triples in this work. [9] uses an existing a structured dataset describing car instances. The dataset was modelled and transformed into RDF triples by the authors using Hepp's Vehicle Sales Ontology[18].

Number of queries and size of queries. Third and fourth columns of Table 7 show heterogeneous number of queries and number of triples patterns used for the experimental evaluations. First, we remark that LUBM queries used in the experiments were different. Each concerned work defined different queries depending on their goals. CADER [7] was evaluated with the greatest number of queries but we recall that the query relaxation consists in dropping failing triple patterns (MFS). The number of triple patters goes from 2 to 15. [4] experimented with five subject star-shaped queries having from three to four triple patterns. [8] and [10] used seven LUBM queries. [8] used queries with a maximum of five triple patterns and one MFS. The maximum number of executed relaxed queries is 8, 9, and 16 queries for k equals to 10, 50, and 150, respectively. [10] defined queries involving until 15 triple patterns and up to 4 MFSs (k was set to 50 answers). This number of triple patterns lead to more than 1000 executed queries. O-MBS and F-MBS reduce the necessary number of queries to be executed to about 80 queries. FLiQue [13] was experimented with 32 federated queries with two up to twelve triple patterns. At most, 69 relaxed queries were generated for the queries needing relaxation. Nevertheless, at most 3 queries were executed for every initial query. We recall that these benchmark queries contain lots of variables and very few classes in objects. Thus the relaxation opportunities are mostly concentrated in the properties.

[7, 9] focus on studying the performance in terms of the quality of results. Thus, the number of triple patterns was not important (one to four triple patterns). [7] defined 40 evaluation queries for each dataset. Top-5 relaxed queries were generated for each evaluation query. [9] defined three vague queries run against the vehicles dataset where five relaxations were also generated for each query.

The clustering-based approach [11] defined sets of small queries (up to 21 triple patterns) and large queries (up to 1505 triple patterns) for the experiments. Small queries were tested to study the efficiency of the

[18] http://www.heppnetz.de/ontologies/vso/ns

proposed algortihms in different execution modes (limit to relaxation distance and computation time). Large queries were used to prove the efficiency of the proposed algorithms in relaxing queries with thousands of triple patterns in a matter of minutes (thanks to the efficiency of similarity search). When large queries (having triple patterns up to 248) were tested over the MONDIAL dataset, a maximum of 117 relaxed queries were computed in less than 270 seconds. However, for queries (having triple patterns up to 1505) tested over LUBM, a maximum of 52 relaxed queries were computed in less than 540 seconds.

Comparision to other works. Due to the lack of availability of source codes and experimental prototypes, it is not common to find experimental evaluations among the state-of-the-art works. CADER was compared experimentally with algorithms proposed in [23] (LBA and MBA algorithms). The experiments proved a better behaviour of CADER in terms of execution time regardless the shape of queries. The algorithms proposed in [10], were compared as a whole with the BFSR algorithm [8]. Experiments revealed that BFSR executes more queries until finding k answers than the proposed algorithms in [10]. For queries with more than ten triple patterns, more than 1000 queries were executed for BFSR, while only about 80 queries were executed in the case of the algorithms based on MFS with k set to 50.

Source code availability. It is hard to reproduce the experimentally analysed works due to the lack of availability of prototypes and source codes. There exist only source codes for [10][19], [13][20], and [11][21].

5.4. General analysis

Several conclusions can be drawn from our analysis and the information shown in the comparative Tables 5-7.

In general, query relaxation addresses applications where obtaining empty or few answers are unsatisfactory. Almost all proposed approaches can deal with star, chain, and composite queries. So, all of them can propose resources similar to those asked by the original query based on the provided descriptions. [2, 3, 5] demonstrated that property path queries can be relaxed. Property path queries provide a concise way to write complex navigational queries over RDF graphs. [13] also showed that even federated queries can be relaxed. Experimented federated queries were of type star, chain, and composite. Their particularity is that the knowledge can be obtained from multiple sources. Thus, federated queries allow the integration of distant datasets.

Concerning relaxation, ontology-based relaxation is an online process. Therefore, even if the relaxation possibilities can be huge, O-MBS and BR ([8, 10]) showed their applicability in a real online context thanks to their optimized strategies to prune the query relaxation lattice. Relaxation based on the similarity of instances needs, in general, an initialization phase that is an offline process. That is because techniques to define similarities are data-type dependent, and some similarity functions are costly. Similarity functions may involve processing external information (canonical representations, dictionaries, similarity tables, etc.), but also processing statistical models (NLP and machine learning techniques). This initialization phase should be done periodically for datasets that change over time. While for ontology-based relaxation, changes in the dataset have little impact on the computation of relaxed queries. Only summaries have to be updated to calculate precisely the similarity of queries. For a federated environment, the indexes allowing to discard empty joins between distant sources must also be updated.

Highlighting one approach based on the similarity of instances is hard because they were tested over different environments. Nevertheless, the clustering-based relaxation work stands out because it does not address specific queries, can relax queries with a high number of joins, and has good behavior with multi-valued properties. Moreover, [11] is the only proposal that can use both ontology-based relaxation and similarity of instances (based on symbolic similarity). A drawback of this approach is its bottom-up navigation of the relaxation lattice that is hard to scale over high-sized datasets.

As we will see in the next part of this survey, none of the analysed proposals can deal conveniently with RDF reification.

[19] http://www.lias-lab.fr/forge/projects/qars

[20] https://github.com/benj-moreau/FLiQue

[21] OCaml https://bitbucket.org/sebferre/sewelis/src/master/, and Java https://gitlab.inria.fr/hayats/CONNOR/-/tree/master

6. Statement-based RDF reification

Within the Semantic Web community, the reification methods have been the subject of many discussions and works for several years. Various approaches have been proposed to represent metadata. Some works analysed and evaluated several representation models for reification based on different criteria [30–33]. Reified RDF triples can have important impact over current relaxation works. The goal of this section is to study the compatibility of the reification models with the current relaxation works and the challenges that can be encountered. We focus on statement-level metadata, i.e., reification at the granularity of an RDF triple.

This section is organized as follows. Section 6.1 describes five popular reification approaches explaining their syntax to define RDF triples and SPARQL queries. Section 6.2 analyses these approaches based on various criteria. Section 6.3 discusses the existing experimental analysis of reification models. Finally, Section 6.4 sheds the lights on the main impact and consequences of applying relaxation approaches on reified triples.

6.1. Reification models

In this work, we analyse five representative reification models, namely, standard reification [34], named graphs [35], n-ary relations [30], singleton properties [36], and RDF-Star [37].

Each approach, in practice, generates a different number of RDF triples, distinct graph representations, and different query syntaxes. These differences have an impact on the storage volume, the query execution time, the bulk load, and the SPARQL integration.

To shed the lights on the main differences among the analysed reification models and to compare their advantages and drawbacks, we use the same running example. In our example, facts represent educational resources (ER) associated to some related topics using the attribute dc:subject. For each RDF statement, metadata about two numeric metrics are considered: a page rank score (un:pageRank) and a cosine similarity (un:cosineSimilarity). Our example query, searches for ERs talking about :Query_Language.

The graph representation of the five reification models we analyse are shown in Figure 4. Listing 1 shows the syntax of the RDF triples of every reification approach. Turtle syntax is used for standard reification, singleton property, and n-ary relations. For named graphs and RDF-star models, TriG and Turtle-star syntaxes are used respectively. Listing 2 shows the SPARQL query that corresponds to each reification model. Two filter expressions over the metadata values are added to specify the range of values.

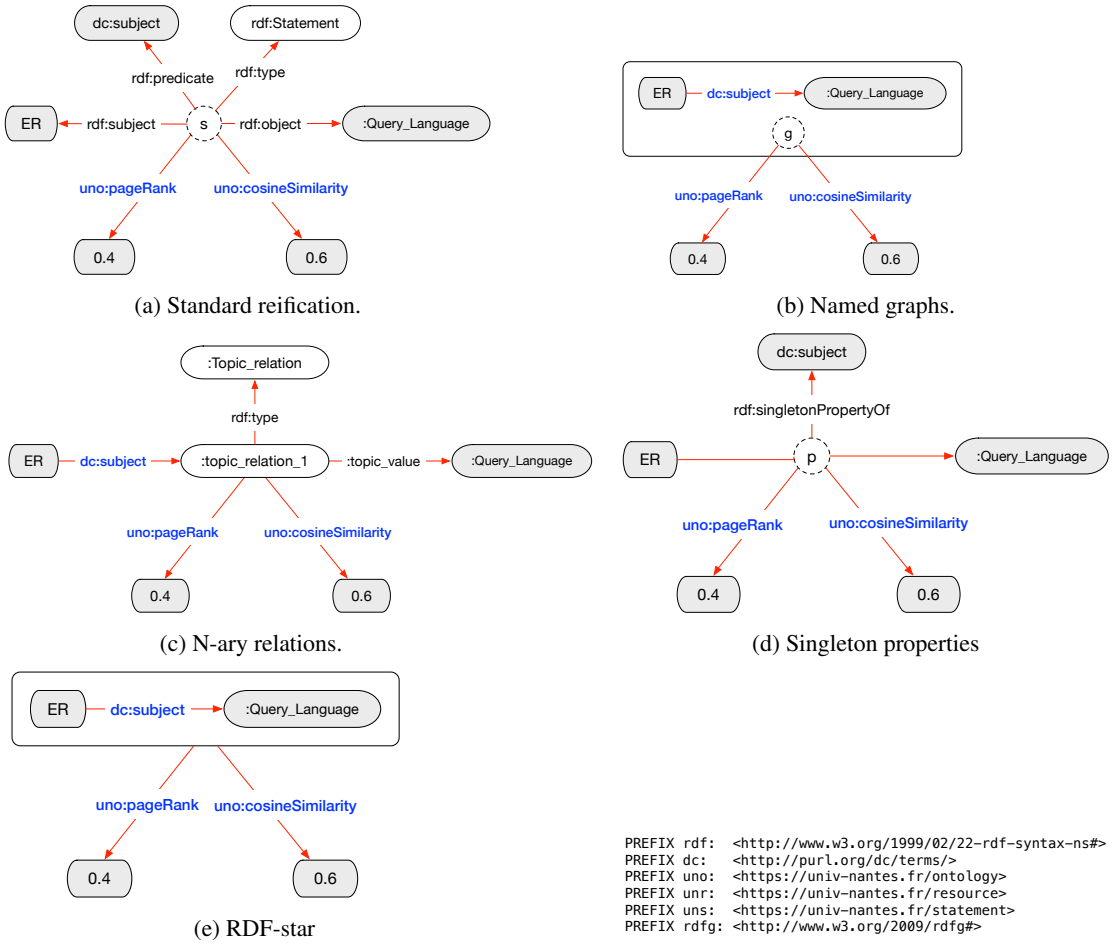
6.1.1. Standard reification

The standard reification method has been proposed within RDF primer standardised by W3C [34][22]. This method relies on the idea of refering to statement-level metadata using resources with no URIs, i.e., blank nodes. These blank nodes are used to represent RDF statements where every statement is defined by three parts according to their roles (rdf:subject, rdf:predicate, and rdf:object). In this method, an identifier is declared to refer to every unique RDF statement.

Figure 4a shows the graph representation of a reified statement using standard reification. A node "s" represents the reified statement identified by id "S100". This node is linked to every part of this statement by edges representing the role of each. The two score values are then linked to the same resource defined for this statement. Hence, the size of complexity of this approach is 4+n, where n is the number of annotations. RDF triples of this example applying standard reification is shown in the Listing 1a. Six triples are necessary for representing the fact and its metadata (in our example we have two annotations). Three triples are added to specify the role of every part of the RDF statement, one triple is used to specify the type of this statement, and two triples to link this statement with the additional metadata.

Listing 2a shows an example of a SPARQL query including standard reification. This SPARQL query contains also four triple patterns that describe the RDF statement and two triple patterns that fetch the metadata values using their related relations.

[22]https://www.w3.org/TR/rdf-primer/#reification



(a) Standard reification. (b) Named graphs. (c) N-ary relations. (d) Singleton properties (e) RDF-star

Fig. 4. Illustration of reification models with graph representations.

6.1.2. Named graphs

Carrol et al. [35] extended the RDF data model to cover RDF graphs nameable by URIs. Named graphs were proposed to describe graphs by addressing a represention of provenance information.[23] This approach considers a set of pairs of the form (g, n), where g is an RDF graph and n is an IRI (or a blank node in some cases, or can even be omitted for a default graph). A unique identifier is used for every graph. One RDF graph is defined for the statement(s) to be annotated, and the annotations are defined into another RDF graph.

Figure 4b illustrates the idea of the named graphs where a single statement exists inside a rectangular shape that represents an RDF graph denoted by g. The annotations are directly linked to this graph. Listing 1b shows the syntax used in named graphs. Each statement to be annotated is defined in one graph. All annotations can then be defined in another graph. In our example, g-100 contains the triple to be annotated and g-101 contains all the annotations. The IRI of the graph corresponding to the statement to be annotated is used as subject in the corresponding annotations.

Listing 2b shows an example of a SPARQL query using named graphs. This approach adds few triple patterns to a query, i.e., three triple patterns. One triple pattern is added for annotated statement (included inside the curly brackets of the Graph statement), and two triple patterns for the metadata values.

[23] https://www.w3.org/TR/rdf11-concepts/

```
uns:S100  rdf:type rdf:Statement ;
          rdf:subject unr:er ;
          rdf:predicate dc:subject ;
          rdf:object :Query_Language ;
          uno:cosineSimilarity "0.6" ;
          uno:pageRank "0.4" .
```

(a) Standard reification in Turtle.

```
<g-100> { unr:er dc:subject :Query_Language . }
<g-101> { <g-100> uno:cosineSimilarity "0.6" ;
                  uno:pageRank "0.4" .
        }
```

(b) Named graphs in TriG.

```
:Topic_relation    rdf:type owl:Class .
unr:er dc:subject  :topic_relation_1 .
:topic_relation_1  rdf:type :Topic_relation ;
                   :topic_value :Query_Language ;
                   uno:cosineSimilarity "0.6" ;
                   uno:pageRank "0.4" .
```

(c) N-ary relations in Turtle.

```
unr:er  <p-200> :Query_Language .
<p-200>  rdf:singletonPropertyOf dc:subject;
         uno:cosineSimilarity "0.6" ;
         uno:pageRank "0.4" .
```

(d) Singleton properties in Turtle.

```
<< unr:er dc:subject :Query_Language >> uno:cosineSimilarity "0.6" ;
                                        uno:pageRank "0.4" .
```

(e) RDF-Star in Turtle-star.

Listing 1. Examples of RDF triples in different reification approaches.

6.1.3. N-ary relations

A fact is a triple linking two resources through a relation or property. The n-ary relation model [38][24] was proposed for relations that link a resource to more than just one resource or value. This approach proposes to use classes instead of properties to do that. Thus, a new class is created to represent the n-ary relation. A relationship as an instance of this class is created and used as a resource of a subject-predicate pair. The instance of a relation can be a blank node.

Figure 4c illustrates the n-ary relations model. In this figure, the new class created is named Topic_relation. The node topic_relation_1, an instance of the new class, is replaced in the position of the object (instead of :Query_Language) of the fact. This node is directly linked to the metadata values. Hence, every reified statement will be identified uniquely by an instance of a new class. While the object node :Query_Language is now linked to

[24]http://www.w3.org/TR/swbp-n-aryRelations

```
SELECT ?er  WHERE {
  ?s     rdf:type rdf:Statement ;
         rdf:subject ?er ;
         rdf:object :Query_Language ;
         rdf:predicate dc:subject ;
         uno:cosineSimilarity ?c ;
         uno:pageRank ?pr .
         FILTER (?c  > 0.6)
         FILTER (?pr  > 0.4)
}
```

(a) Standard reification.

```
SELECT ?er  WHERE {
   GRAPH  ?g {
              ?er dc:subject  :Query_Language .
            }
   GRAPH  ?m {
               ?g  uno:cosineSimilarity ?c ;
                   uno:pageRank ?pr .
            }
         FILTER (?c  > 0.6)
         FILTER (?pr  > 0.4)
  }
```

(b) Named graphs.

```
SELECT ?er  WHERE {
    ?er     dc:subject   ?t .
    ?t      :topic_value  :Query_Language ;
            uno:cosineSimilarity ?c ;
            uno:pageRank ?pr .
    FILTER (?c  > 0.6)
    FILTER (?pr  > 0.4)
 }
```

(c) N-ary relations.

```
SELECT ?er  WHERE {
    ?er      ?p  :Query_Language .
    ?p       rdf:singletonPropertyOf  dc:subject ;
             uno:cosineSimilarity ?c ;
             uno:pageRank ?pr .
    FILTER (?c  > 0.6)
    FILTER (?pr  > 0.4)
  }
```

(d) Singleton properties.

```
SELECT ?er  WHERE {
     << ?er dc:subject :Query_Language >> uno:cosineSimilarity ?c ;
                                          uno:pageRank ?pr .
      FILTER (?c  > 0.6)
      FILTER (?pr  > 0.4)
     }
```

(e) RDF-Star.

Listing 2. Example of SPARQL queries using different reification approaches.

	# Triples	# triple patterns	# overhead variables
Standard reification	5	5	1
N-ary	4	3	2
Singleton properties	3	3	1
Named graphs	2	2	1
RDF-star	1	1	0

Table 8
Comparison of reification approaches.

this instance node using a predicate :topic_value. The syntax of the RDF triples are shown in Listing 1c. Five triples are needed for our example in the n-ary approach. In the first triple, the object is replaced with the instance of the new class. The second triple specifies the type of this instance (new created class). In addition, :Query_Language is now linked with this instance using a new predicate. Finally, two triples state the metadata values.

Listing 2c shows an example of SPARQL query for this approach. Three triple patterns must be added for the query, two triple patterns representing the RDF statement and one triple pattern representing the annotation of this statement.

6.1.4. Singleton properties

The singleton property [36] proposes to create a unique property for every triple that has associated metadata. This unique property can be seen as a predicate resource IRI. Figure 4d shows the graph representation of this model. In this representation, a new node is created to represent the new property. This property is connected directly to the original property via the proposed property *singletonPropertyOf*. The same property is used for all metadata associated to a statement. Example of RDF triples is represented in Listing 1d. Four triples are used for representing metadata using this approach. In this example, an instance of a new property is created and identified using a property id p-200. This identifier is directly linked with our two metadata.

An example of SPARQL query using this approach is shown in Listing 2d. In this approach, three triple patterns are needed. The initial predicate in the statement is replaced with a variable that refers to an instance of a new property (Line 2). A triple pattern is added to link this variable with the original predicate via rdf:singletonPropertyOf (Line 3). The same variable is also used to refer to the metadata.

6.1.5. RDF-star

Recently, [37][25] proposed RDF-star and SPARQL-star, as an extension of RDF and SPARQL to allow graph nestings. The aim is to simplify the represention of reified statements. Therefore, this extension gives the ability to set graphs into graphs recursively. This approach does not need declaring edge identifiers that will be linked with the metadata. This model is depicted in the graph representation in Figure 4e. The fact to be annotated is inside a rectagnular box and this box is directly linked to the annotations. Precisely, RDF-star allows triples to have other triples in subjects or objects by using double angle brackets $\ll \gg$. Hence, every reified statement can be interpreted as a single RDF triple (see Listing 1e).

SPARQL-star allows to bind a triple pattern enclosed in angle brackets as subject or object on other triple pattern. An example is shown in Listing 2e. This syntax shows that only one triple pattern is needed in a query to search in a reified statement.

RDF-star and SPARQL-star are still not a W3C recommendation. An RDF-star working group[26] was created to work on extending the RDF and SPARQL recommendations.

6.2. Syntax analysis of reification models

The described reification models differ in various criteria such as the number of triples, human understandability, flexibility, and syntax support.

[25]https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html
[26]https://www.w3.org/2022/08/rdf-star-wg-charter/

Number of triples. Second column of Table 8 shows the number of triples that are necesary to describe a statement with one annotation. Standard reification is the most costly approach since it needs five triples for one reified statement. N-ary model needs four triples plus a new class declaration. This class declaration is done once by statement whatever the number of annotations. The singleton models needs three triples. Named graphs and RDF-star are the most compact models needing two and one triples respectively. One advantage of named graphs is that reification can be defined not only at statement level but also for a group of triples or even a dataset. Only this model allows these different granularities for annotations.

Human understandability. Standard reification, singleton properties, and n-ary relations can be considered as hardly human understandable because the reified statement is split in several forms. Consequently, it is not possible to get the reified statement at once in a simple maner. Named graphs is more clear since it does not break the direct links between the terms of the reified statement. RDF-star was proposed to simplify statement-level annotations so it gives a clear representation of statement and their annotations. It does not involve refactoring the statement, declaring new predicates or new classes. This ensures an easy human understandability.

Flexibility. All these reification models are flexible with respect to adding new annotations to an already reified statement. Adding new annotations requires adding only one additional triple for every approach. Concerning multi-valued properties (i.e., annotations having a subject-predicate pair with several objects), all models support it.

Query facilities. We consider the number of triple patterns and the number of overhead variables necessary to query a single triple with its metadata as indicators of query facilities. Columns three and four of Table 8 show these indicators. The number of necessary triple patterns follows the ranking of the number of triples (column two). Standard reification is the model that needs the most number of triple patterns to be declared in a SPARQL query and RDF-star is the model that needs the least. From the perspective of the overhead variables, all models need only one extra variable except RDF-star that needs no extra variable to query the statement and its metadata. According to Table 8, RDF-star is the simplest approach concerning query facilities.

Syntax support. Standard reification, n-ary relations, and singleton properties are conform to the core RDF model proposed in 2004. Named graphs represent an extension to the triple RDF model and is part of the standard RDF1.1 published in 2014. RDF-star proposes to extend the RDF specification. All models are supported in the SPARQL standard, except for RDF-star that proposes SPARQL-star as query language. However, nowadays several RDF stores support the implementation of RDF-star and SPARQL-star such as Apache Jena, Stardog, RDF4j, BlazeGraph, AllegroGraph, etc.[27]

6.3. Existing experimental analysis of reification models

Several works studied different reification methods and compared them according to several criteria. In this section we review some of them.

[30] focused on Wikidata and its representation in RDF using reification based on n-ary relations, standard reification, singleton properties and named graphs. Authors compared these models over five triplestores: 4store, Blazegraph, GraphDB, Jena TDB, and Virtuoso. Their performance were measured based on 14 queries. Their results suggested that the singleton properties model was hardly supported but no other model was an outright winner. Concerning query performance, Virtuoso was the best followed by GraphDB and Blazegraph.

[32] realized an extensive analysis of standard reification, named graphs, n-ary relations, singleton properties, companion properties (proposed in that paper) and RDF-star in its early stages. Experiments used Wikidata and DBpedia datasets on the triplestores Blazegraph, Stardog and Virtuoso. As DBpedia does not have singificant metadata, authors build a dataset with the Wikipedia revision history focusing on a company dataset. The scenario described in [31] was extended to be used with RDF-star. Authors used 31 query templates and generated 300 query instantiations by template. For the DBpedia dataset, authors used 6 query templates and choosed randomly a set of 40 instances from their DBpedia evaluation dataset to populate the templates. The experiments show that when

[27] https://w3c.github.io/rdf-star/implementations.html

the granularity of metadata is not by statement, companion properties and named graphs outperform. Concerning statement-level metadata, while standard reification results in the highest number of triples, it consumes the least storage in the database files and named graphs the most. This is because additional index structures for the graph identifiers are maintained. Concerning query performance, the obtained results indicate that metadata characteristics have an impact on the ranking of the reification models. Named graphs and RDF-star support queries against meta-metadata much better than the other models. In general, RDF-star can compete with named graphs, if the metadata is on statement level. In addition, experimental results show that the ranking of reification models differs between data-only and mixed query scenarios. Moreover, named graphs and RDF-star offer the best trade-off for both, mixed and data query workloads

[39] used three simple counting queries to analyse the internal representations of RDF-star in Stardog, Blazegraph and ExecuteSPARQLStar.[28] Experiments showed the divergence of the implementations of RDF-star when dealing with nested RDF-star statements. Blazegraph and ExecuteSPARQLStar behave similarly but Stardog was not able to deal correctly with nested RDF-star statements. That is because Stardog flattens the nested statements.

[33] proposes a benchmark (dataset and set of queries) to analyse reification models. To illustrate the utility of the benchmark, authors analysed querying performance, storage efficiency and usability on the Stardog triplestore using three reification models: standard reification, singleton properties and RDF-star. Authors used the Biomedical Knowledge Repository (BKR) dataset[29] in order to make their results comparable with [36] that compares singleton property against standard reification. Twelve queries are used to evaluate performance. Five of these queries were proposed in this work to focus on SPARQL-star. Experimental results suggest that singleton property seems to have the worst performance. Probably because of the high number of unique properties and because indexes are usually optimised for a high number of distinct subjects or objects and not for a high number of distinct predicates. Authors also observed that for simple queries standard reification performs better than RDF-star. For complex queries, clearly, RDF-star outperforms standard reification.

6.4. Effect of query relaxation strategies on reified triples

Query relaxation techniques will behave strangely in presence of reification.

Query size. Querying data and its metadata need, in general, queries with more triple patterns and more joins. As the number of triple patterns increases, the complexity of the computation of relaxed queries increases. RDF-star and named graphs are the most compact models and could be the most efficient during query relaxation from the complexity point of view. Unfortunately, there is no relaxing approach that uses either SPARQL-star or named graph queries.

Syntax support. Queries for standard reification, n-ary relations and singleton properties can be relaxed with existing relaxation approaches. However, current works cannot relax SPARQL-star and named graphs queries because of their nested approach, i.e., the subject or the object can be another triple pattern or a graph.

Relaxation over annotations. Relaxation based on ontologies over queries including metadata constraints will apply relaxation over the predicates related to the metadata or the metadata value itself. Ontology-based techniques will replace a property with its superproperty. However, the property relaxation will not have any practical effect because relaxing predicates will only substitute the metadata relations without considering their values. In addition, in general, the ontology hierarchy of properties used in annotations is not very rich. In simple relaxation, a variable would replace the instance or the constant in the subject or the object of a metadata triple pattern. This relaxation would have a better effect than the property relaxation. If relaxation is based on the similarity of instances, the annotation values will be replaced with similar or closer values. Depending on the type of the metada value this relaxation could have a better effect than the the property and simple relaxations.

[28] https://github.com/RDFstar/RDFstarTools
[29] https://zenodo.org/record/3894746#.ZAtF0S_pNpQ

Metadata type. Metadata type (numeric, string, date, URIs, etc.) is relevant for the relaxation based on the similarity of instances. The similarity functions to estimate the distance of resources is data-type depend. Estimating the distance of all the resources in a graph is a costly process that is frequently done offline. Thus, data types used in annotations may have an impact on the initialisation phase for techniques based on the similarity of instances.

Dataset size. Conversely to ontology-based relaxation, the relaxation approaches based on similarity of instances need an offline initialization phase before the online query processing. The initialization phase can be costly in time and its complexity depends on the size of the database. Except for named graphs and RDF-star, other models need several triples to include statement-based annotations. This volume will directly impact the first phase since loading and analysing all these triples would require additional time. Besides, the shape of RDF graph depends on the reification model. Thus, approaches like the one based on statistical language model relaxation [7] or the clustering-based relaxation [11] would be seriously impacted. On the other side, ontology-based relaxation approaches are independent of the size of the dataset, i.e., they depend on the class and property hierarchies of the ontology. Except for n-ary relations where one class definition is necessary by annotated statement, the performance of reification over ontology-based approaches would be insignificant.

7. Challenges and open issues

Applying query relaxation over queries whose constraints concern data and metadata (statement-level annotations) opens several issues. Current relaxation techniques are not proposed for querying metadata. (i) Logical relaxation over properties of metadata triple-patterns (with their superproperties) has in general no sense. (ii) Metadata values are not taken into account in the query ranking function that allows gathering the *k-relevant* answers closest to the original query. (iii) Current relaxation techniques are defined for equality of values but metadata constraints can include intervals of values in a range. (iv) Syntaxes to represent RDF reification and to query reified triples induce strange behavior in current query relaxation approaches.

Hence, we consider that new query relaxation contributions should be proposed to deal with queries querying data and metadata. A query rewritting process is necessary to distinguish the query constraints (which triple patterns concern data and which ones metadata) because relaxing metadata triple-patterns has not the same goal as relaxing other triple patterns. Both, ontology-based relaxation and similarity of instances, are necessary to query data and metadata. Depending on the application goals, these techniques may be combined. Querying data and metadata increases the query size which leads to the increase of the query relaxation lattice. Thus, optimal methods to prune this relaxation lattice should be used. Metadata values should play a role in the query ranking strategies that allow pruning the relaxation lattice but also gathering the *k-relevant* answers closest to the original query. Thus, new functions to calculate the similarity of queries should be proposed. Finally, relaxing metadata triple-patterns should take into account the type of metadata values. For the similarity of instances, the potential number of metadata types will be challenging. The number of similarity functions necessary to take into account all metadata types can be important which will add complexity to the similarity of instance approaches.

8. Conclusion

Applications querying reified triples may face the problem of empty or insufficient answers. Query relaxation approaches have been proposed to solve this problem but none of them is appropriate for metadata triple-patterns. In this paper, we provided an overview and comparative analysis of existing contributions focusing on SPARQL query relaxation. We also analysed and compared the syntaxes of some relevant reification models. Then, we underlined the potential effects of query relaxation approaches over reified triples. This survey has revealed that at the moment, no query relaxation solution deals with RDF triples and their annotations. Therefore, we pointed out some challenges and open issues in relaxing SPARQL queries under the lens of RDF reification, which we hope will open the doors to new inspiring contributions.

9. Acknowledgments

This work has received a French government support granted to the Labex Cominlabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.

References

[1] T. Gaasterland, P. Godfrey and J. Minker, Relaxation as a platform for cooperative answering, *Journal of Intelligent Information Systems* **1**(3/4) (1992), 293–321.
[2] C.A. Hurtado, A. Poulovassilis and P.T. Wood, A relaxed approach to RDF querying, in: *International Semantic Web Conference (ISWC)*, Vol. 4273, Springer, 2006, pp. 314–328.
[3] C.A. Hurtado, A. Poulovassilis and P.T. Wood, Query relaxation in RDF, *Journal on Data Semantics X* **4900** (2008), 31–61.
[4] H. Huang, C. Liu and X. Zhou, Computing relaxed answers on RDF databases, in: *International Conference on Web Information Systems Engineering (WISE)*, Vol. 5175, Springer, 2008, pp. 163–175.
[5] A. Poulovassilis and P.T. Wood, Combining approximation and relaxation in semantic web path queries, in: *International Semantic Web Conference (ISWC)*, Springer, 2010, pp. 631–646.
[6] H. Huang and C. Liu, Query relaxation for star queries on RDF, in: *International Conference on Web Information Systems Engineering (WEBIST)*, Vol. 6488, Springer, 2010, pp. 376—389.
[7] S. Elbassuoni, M. Ramanath and G. Weikum, Query relaxation for entity-relationship search, in: *Extended Semantic Web Conference (ESWC)*, Vol. 6644, Springer, 2011, pp. 62–76.
[8] H. Huang, C. Liu and X. Zhou, Approximating query answering on RDF databases, *World Wide Web (WWW)* **15** (2012), 89–114.
[9] A. Hogan, M. Mellotte, G. Powell and D. Stampouli, Towards fuzzy query relaxation for RDF, in: *Extended Semantic Web Conference (ESWC)*, Vol. 7295, Springer, 2012, pp. 687–702.
[10] G. Fokou, S. Jean, A. Hadjali and M. Baron, RDF query relaxation strategies based on failure causes, in: *European Semantic Web Conference (ESWC)*, Vol. 9678, Springer, 2016, pp. 439–454.
[11] S. Ferré, Answers partitioning and lazy joins for efficient query relaxation and application to similarity search, in: *Extended Semantic Web Conference (ESWC)*, Vol. 10843, Springer, 2018, pp. 209–224.
[12] W. Mebrek, B. Raddaoui and M. Albilani, On relaxing failing queries over RDF databases, in: *2019 IEEE International Conference on Big Data (IEEE Big Data)*, IEEE, 2019, pp. 115–124.
[13] B. Moreau and P. Serrano-Alvarado, Ensuring license compliance in linked data with query relaxation, in: *Transactions on Large-Scale Data-and Knowledge-Centered Systems (TLDKS)*, Vol. 12920, Springer, 2021, pp. 97–129.
[14] P. Dolog, H. Stuckenschmidt and H. Wache, Robust query processing for personalized information access on the semantic web, in: *International Conference on Flexible Query Answering Systems (FQAS)*, Vol. 4027, Springer, 2006, pp. 343–355.
[15] J. Heer, M. Agrawala and W. Willett, Generalized selection via interactive query relaxation, in: *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI)*, 2008, pp. 959–968.
[16] I. Muslea, Machine learning for online query relaxation, in: *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2004, pp. 246—255.
[17] D. Mottin, A. Marascu, S. Basu Roy, G. Das, T. Palpanas and Y. Velegrakis, IQR: an interactive query relaxation system for the empty-answer problem, in: *Proceedings of the ACM SIGMOD international conference on Management of Data*, 2014, pp. 1095—1098.
[18] P. Resnik, Using information content to evaluate semantic similarity in a taxonomy, in: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 1, 1995, pp. 448—453.
[19] S. Ross, *A first course in probability*, Macmillan, 1976.
[20] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song and D. Zhao, Semantic SPARQL similarity search over RDF knowledge graphs, *Proceedings of the VLDB Endowment* **9**(11) (2016), 840–851.
[21] G.F. Cooper and E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, *Machine learning* **9** (1992), 309–347.
[22] Y. Guo, Z. Pan and J. Heflin, LUBM: A benchmark for OWL knowledge base systems, *Journal of Web Semantics (JWS)* **3** (2005), 158–182.
[23] G. Fokou, S. Jean, A. Hadjali and M. Baron, Cooperative techniques for SPARQL query relaxation in RDF databases, in: *European Semantic Web Conference (ESWC)*, Vol. 7295, Springer, 2015, pp. 687–702.
[24] B. Moreau, P. Serrano-Alvarado, M. Perrin and E. Desmontils, Modelling the compatibility of licenses, in: *The Semantic Web: 16th International Conference of European Semantic Web Conference (ESWC)*, Vol. 11503, Springer, 2019, pp. 255–269.
[25] M. Saleem, A. Potocki, T. Soru, O. Hartig and A.-C.N. Ngomo, CostFed: Cost-based query optimization for SPARQL endpoint federation, *Procedia Computer Science* **137** (2018), 163–174.
[26] M. Saleem, A. Hasnain and A.-C.N. Ngomo, LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation, *Journal of Web Semantics (JWS)* **48** (2018), 85–125.
[27] A. Hogan, A. Zimmermann, J. Umbrich, A. Polleres and S. Decker, Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora, *Journal of Web Semantics (JWS)* **10** (2012), 76–110.

[28] B. Ganter and R. Wille, *Formal concept analysis: mathematical foundations*, Springer Science & Business Media, 2012.
[29] Z. Ge, Y. Wang, H. Yan and X. Xu, A learning-based semantic approximate query over RDF knowledge graph, in: *International Conference on Advanced Cloud and Big Data (CBD)*, IEEE, 2018, pp. 135–141.
[30] D. Hernández, A. Hogan and M. Krötzsch, Reifying RDF: What works well with wikidata?, *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems(SSWS)* **1457** (2015), 32–47.
[31] D. Hernández, A. Hogan, C. Riveros, C. Rojas and E. Zerega, Querying wikidata: Comparing SPARQL, relational and graph databases, in: *International Semantic Web Conference (ISWC)*, Vol. 9982, Springer, 2016, pp. 88–103.
[32] J. Frey, K. Müller, S. Hellmann, E. Rahm and M.-E. Vidal, Evaluation of metadata representations in RDF stores, *Semantic Web Journal (SWJ)* **10** (2017), 205—229.
[33] F. Orlandi, D. Graux and D. O'Sullivan, Benchmarking RDF metadata representations: reification, singleton property and RDF, in: *IEEE International Conference on Semantic Computing (ICSC)*, 2021, pp. 233–240.
[34] F. Manola, E. Miller, B. McBride et al., RDF primer, *W3C recommendation* **10**(1–107) (2004), 6.
[35] J.J. Carroll, C. Bizer, P. Hayes and P. Stickler, Named graphs, *Journal of Web Semantics (JWS)* **3**(4) (2005), 247–267.
[36] V. Nguyen, O. Bodenreider and A. Sheth, Don't like RDF reification? Making statements about statements using singleton property, in: *Proceedings of the 23rd international conference on World Wide Web (WWW)*, 2014, pp. 759–770.
[37] O. Hartig, Foundations of RDF* and SPARQL*:(An alternative approach to statement-level metadata in RDF), in: *International Workshop on Foundations of Data Management and the Web (AMW)*, Vol. 1912, Juan Reutter, Divesh Srivastava, 2017.
[38] N. Noy, A. Rector, P. Hayes and C. Welty, Defining n-ary relations on the semantic web, *W3C working group note* (2006).
[39] F. Orlandi, D. Graux and D. O'Sullivan, How many stars do you see in this constellation?, in: *European Semantic Web Conference (ESWC), poster demo*, Vol. 12124, Springer, 2020, pp. 175—180.