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. Query relaxation has been proposed 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 querying RDF datasets generally relax the SPARQL 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., RDF reification. In this survey, we overview SPARQL query relaxation works with a particular focus on issues and challenges posed by representative RDF reification models, namely, standard reification, named graphs, n-ary relations, singleton properties, and RDF-Star.

Keywords: SPARQL query relaxation, RDF datasets, RDF reification models, statement-level annotations, 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, 2]. Deductive databases not only store explicit information in relational databases but they also store rules (Datalog or Prolog rules) that enable inferences based on the stored data. The original proposed idea, is a relaxation method to expand the scope of the query dynamically by relaxing the constraints in the query (predicates, constants, and breaking join dependencies). The goal is to generalize the user query to produce more answers by exploitng the hierarchical structures defined in a is-a taxonomy.

The evolution of the Web, to the Web of Data lead to the development of data graph structures (RDF[1] graphs) constrained by complex ontologies using expresive languages like RDFS[2] and OWL[3]. Unlike is-a taxonomies, ontologies allow defining complex relationships like the properties and attributes associated with classes, providing a detailed characterization of entities. Moreover, ontologies enable the specification of the domain (the class to which a property belongs) and range (the class that the property points to) for more precise modeling. Several

[*] Corresponding author. E-mail: ginwa.fakih@univ-nantes.fr.
[1] https://www.w3.org/TR/rdf11-primer/
[2] https://www.w3.org/TR/rdf11-schema/
[3] https://www.w3.org/TR/owl-overview/

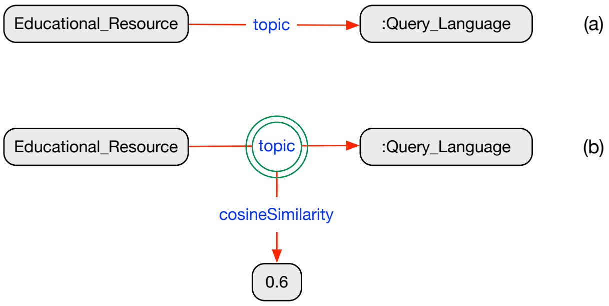


Fig. 1. Representation of context on labeled edge

languages are commonly used for querying RDF dataset, e.g., SPARQL[4], RDQL[5], etc. However, SPARQL remains the most widely adopted and standardized language for querying RDF graphs.

In this survey, we focus on SPARQL queries over RDF triples. SPARQL allows defining queries with triple patterns over which conjunctions, disjunctions, and optional patterns can be defined. With SPARQL, query issuers can use the OPTIONAL clause 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 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 only the ontology hierarchy or using information content [3] 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.

Different 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 in several ways, 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, trust values, scores, weights, etc. When posing queries searching for annotation values, these values may be compared. RDF 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. There is for instance, standard reification [4], named graphs [5], singleton properties [6], RDF-star [7], etc.

[4] SPARQL 1.1 Query Language W3C Recommendationhttps://www.w3.org/TR/sparql11-query/
[5] https://www.w3.org/submissions/RDQL/

Existing query relaxation approaches have proved their efficiency but statement-level reification can lead to significant challenges for relaxation techniques. For instance, if a query seeks resources with the topic :Query_Language (?er, :topic, :Query_Language), relaxation might replace it with the more general class :Programming_Language. However, without considering the precised context (how much relevant are the returned educational resources), less relevant educational resources could be retrieved. The same drawback would exist if a similarity of instances is used without taking into account reification. Replacing :Query_Language with a variable leads to numerous results, disregarding the topic. Varying SPARQL syntax over reified triples, depending on the reification model, affects query relaxation behavior. Additionally, performance issues may arise due to triplestore implementations and limited support for reification models.

These challenges motivate this survey on SPARQL query relaxation works with a particular focus on issues and challenges over RDF 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 introduces the background by providing important definitions. Section 4 reviews query relaxation strategies and studies their behaviour though a motivating example. Section 5 compares and analyzes relaxation techniques according to several aspects (query shapes, relaxed terms, optimisation strategies, experimental evaluations, etc.) and analyzes their impact when different RDF reification models are used. Finally, Section 6 discusses the main issues and challenges of applying relaxation on SPARQL queries executed over reified triples. Section 7 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[6], DBLP[7], HAL[8], ACM Computing Surveys[9], the Semantic Web Journal[10], Springer[11], and Journal of Web Semantics[12]. 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 analyzed.

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 December 2023. SPARQL became a W3C Candidate Recommendation on 6 April 2006.

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). Our intention is to include works specifically addressing the automatic relaxation of queries over RDF data within the semantic web domain. Hence, papers outside this scope or addressing other types of data structure (relational, XML, NewSQL, NoSQL, etc.) were excluded. Thus, we review 14 approaches proposed in 13 research papers (see Table 1).

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

Year	Reference	Title	Journal/Conference	Authors
2006	[8]	A relaxed approach to RDF querying	ISWC	Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood
2008	[9]	Query relaxation in RDF	Journal on Data Semantics	Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood
2008	[10]	Computing relaxed answers on RDF databases	WISE	Hai Huang, Chengfei Liu, and Xiaofang Zhou
2010	[11]	Combining approximation and relaxation in semantic web path queries	ISWC	Alexandra Poulovassilis, and Peter T. Wood
2010	[12]	Query relaxation for star queries on RDF	WISE	Hai Huang and Chengfei Liu
2011	[13]	Query relaxation for entity-relationship search	ESWC	Shady Elbassouni, Maya Ramanath, and Gerhard Weikum
2012	[14]	Approximating query answering on RDF databases	WWW	Hai Huang, Chengfei Liu, and Xiaofang Zhou
2016	[15]	Towards fuzzy query relaxation for RDF	ESWC	Aidan Hogan, Marc Mellotte, Gavin Powell, and Dafni Stampouli
2016	[16]	RDF query relaxation strategies based on failure causes	ESWC	Geraud Fokou, Stephane Jean, Allel Hadjali, and Mickael Baron
2018	[17]	Answers partitioning and lazy joins for efficient query relaxation and application to similarity search	ESWC	Sebastian Ferre
2019	[18]	On relaxing failing queries over RDF databases	International Conference on Big Data	Wafaa Mebrek, Badran Raddaoui, and Mohamad Albilani.
2021	[19]	Query relaxation for portable brick-based applications	BuildSys	Imane Lahmam Bennani, Anand Krishnan Prakash, Marina Zafiris, Lazlo Paul, Carlos Duarte Roa, Paul Raftery, Marco Pritoni, and Gabe Fierro.
2021	[20]	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.

3. Background

Query relaxation is the task of modifying the query conditions automatically when a query fails to generate sufficient results. 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 based on logical relaxation using ontologies, 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. Before surveying existing relaxation works we introduce the background necessary to understand our analysis. The background presented in this section includes some basic definitions about Semantic Web technologies (Section 3.1), an introduction to SPARQL query relaxation (Section 3.2), and an overview of some RDF reification models (Section 3.3).

3.1. RDF, SPARQL, and RDFS entailment

The Semantic Web is built upon key technologies like RDF (Resource Description Framework), RDFS (RDF Schema), OWL (Web Ontology Language) and SPARQL (SPARQL Protocol and RDF Query Language). These technologies, standardised by the W3C, facilitate the structured representation and efficient retrieval of web-based data. RDF is a standard model for data interchange on the web. While RDFS and OWL are languages designed to represent rich and complex knowledge about things, groups of things, and relations between things. SPARQL is the standard RDF query language and protocol that allows to search, extract, and retrieve relevant information. This section provides an overview of these foundational technologies.

3.1.1. RDF

RDF provides the data model in the form of a graph. It extends the linking structure of the web to use IRIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). An RDF triple is a fact represented as $(s, p, o) \in (\mathrm{I} \cup \mathrm{B}) \times \mathrm{I} \times (\mathrm{I} \cup \mathrm{B} \cup \mathrm{L})$, where I is the set of IRIs, B is the set of blank nodes, and L is the set of literals. An RDF graph G is a set of RDF triples.

Figure 2 shows some RDF triples (Turtle syntax)(on the left) and the corresponding RDF graph (on the right) that represent the knowledge about Maya, a student enrolled in the Semantic Web course teached by Patricia and who knows Peter.

```
@prefix ex:  <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
    ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:Maya        rdf:type       ex:Student  ;
               ex:enrolledIn  ex:Semantic_Web  ;
               foaf:knows     ex:Peter .
               foaf:knows     ex:Patricia .
ex:Patricia    rdf:type       ex:Teacher  ;
               ex:teaches     ex:Semantic_Web  .
```

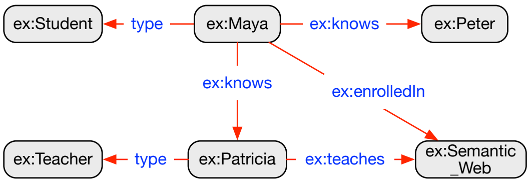


Fig. 2. Example of some RDF triples and the corresponding data graph.

3.1.2. SPARQL

SPARQL is a query language that enables the expression of queries across of RDF data sources. Several types of queries can be defined (SELECT, CONSTRUCT, ASK, and DESCRIBE). In this article we focus on SELECT queries.

The SPARQL semantics is centered on matching graph patterns of the target data graph. The simplest graph pattern is a triple pattern composed of a subject, a predicate, and an object $(s\ p\ o) \in (\mathrm{I} \cup \mathrm{V} \cup \mathrm{B}) \times (\mathrm{I} \cup \mathrm{V}) \times (\mathrm{I} \cup \mathrm{V} \cup \mathrm{L} \cup \mathrm{B})$, where V is a set of variables (each variable is denoted by a question mark) disjoint from the sets I and L. For instance, this triple pattern searches for students enrolled in the Semantic Web course:
`?student ex:enrolledIn ex:Semantic_Web`.

Using the RDF graph of Figure 2, the graph pattern matching will return the mapping: `?student` → `ex:Maya`.

Graph patterns can be combined by conjunction in Basic Graph Patterns (BGP) which combine triples patterns. A BGP can include FILTERs to limit the results of a graph pattern match based on a specified constraint. A Group Graph Pattern (GGP) is delimitted with braces {}. Over GGPs it is possible to define optional or alternative graph patterns. Thus, the SPARQL syntax considers the following operators.

- AND or . allows to define the join operation over a BGP through common variables. All graph patterns are required and if one triple pattern has no binding (i.e., no solution in the data graph) then the query returns no solutions. For instance, this BGP searches for students enrolled in the Semantic Web course and which know Peter `{?student ex:enrolledIn ex:Semantic_Web . ?student foaf:knows ex:Peter}`.
  The graph pattern matching will return the mapping: `?student` → `ex:Maya`.
- FILTER limits the results of a graph pattern match based on a specified constraint. FILTERs eliminate solutions when substituting values into the expression leads to either a false boolean value or an error. For example, this BGP searches for students enrolled in Web courses:
  `{?student ex:enrolledIn ?course FILTER regex(?course "Web", "i"}`.
  The returned mapping is: `?student` → `ex:Maya`.

- OPTIONAL allows to extend the solution of a BGP. If the optional part does not match, it generates no bindings but does not eliminate the solutions. This operator is useful when some data might be useful if available but not strictly required for the query result. For example, this BGP searches for students enrolled in the Semantic Web course, and if available, it returns their mailbox: `{?student ex:enrolledIn ex:Semantic_Web OPTIONAL {?student foaf:mbox ?mail}}` The returned mapping is: `?student` $\rightarrow$ `ex:Maya`.
- UNION allows to define alternative or disjunctive solutions. If multiple alternatives match, it retrieves all these pattern solutions. For instance, this BGP searches for students or teachers:
  `{{?teacher rdf:type ex:Teacher} UNION {?student rdf:type ex:Student}}`.
  The mappings returned are: `?teacher` $\rightarrow$ `ex:Patricia` and `?student` $\rightarrow$ `ex:Maya` .

Queries can be distinguished by the shape of their BGPs into four types: star, chain, composite, and property path.

Star queries. In a star query, all triple patterns share the same variable in the subject or the object. If all triple patterns are centered around the same subject, it is called a subject star-shaped query.

Chain queries. 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.

Composite queries. A composite query is a combination of star and chain queries.

Property path queries. A property path query is a query whose predicate is a regular expression over a set of properties and not only one property. It allows complex paths between nodes instead of just adjacent neighboring nodes.

3.1.3. RDFS entailment

RDFS (RDF Schema) provides basic elements for the description of ontologies. Several RDFS concepts are included in OWL (Web Ontology Language) which is a more expressive language. RDFS constructs allow to define classes and associate properties. Properties define the relations between subject resources and object resources.

Query relaxation is frequently based on the entailment rules of RDFS[13] that we show in Table 2.

- **Subproperty rules** state that:
  Rule (1) defines the properties entailment through the property hierarchy.
  Rule (2) specifies that when a property is inherited from another, if something relates to the superproperty, it is also entailed for the subproperty.
- **Subclass rules** state that:
  Rule (3) defines the classes entailment through the class hierarchy.
  Rule (4) specifies that when a class inherits from another, it is entailed that instances of the subclass are also instances of the superclass.
- **Typing rules** are based on the domain and range of predicates.
  Rule (5) entails the type of a subject in a triple from the domain of the related predicate.
  Rule (6) entails the type of an object in a triple from the range of the related predicate.

3.2. SPARQL query relaxation

The query relaxation explained in this section was proposed in [8] 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.

[13] 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 (s, a, o)	⇒ (s, b, o)
(3) rdfs11	(a, subClass, b) AND (b, subClass, c)	⇒ (a, subClass, c)
(4) rdfs9	(a, subClass, b) AND (s, type, a)	⇒ (s, type, b)
(5) rdfs2	(a, domain, c) AND (s, a, o)	⇒ (s, type, c)
(6) rdfs3	(a, range, d) AND (s, a, o)	⇒ (o, type, d)

Table 2

Subproperty, subclass and type entailment rules of RDFS.

3.2.1. Triple pattern relaxation

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 (?s, a, o) with (?s, type, c) if the triple (a, domain, c) exists in the ontology, or a triple pattern (s, a, ?o) with (?o, type, d) if (a, range, d) exist in the ontology.
- *Triple pattern deletion* removes a triple pattern from the original query.

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 $e' \in \{\prec sp, \prec sc, \prec s\}$.

Figure 3 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 3a, one element of tp can be relaxed once (the object) while in Figure 3b, it can be relaxed twice. In Figure 3c, two terms can be relaxed once (the predicate and the object). The bottom-right corner of Figure 4, 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:

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

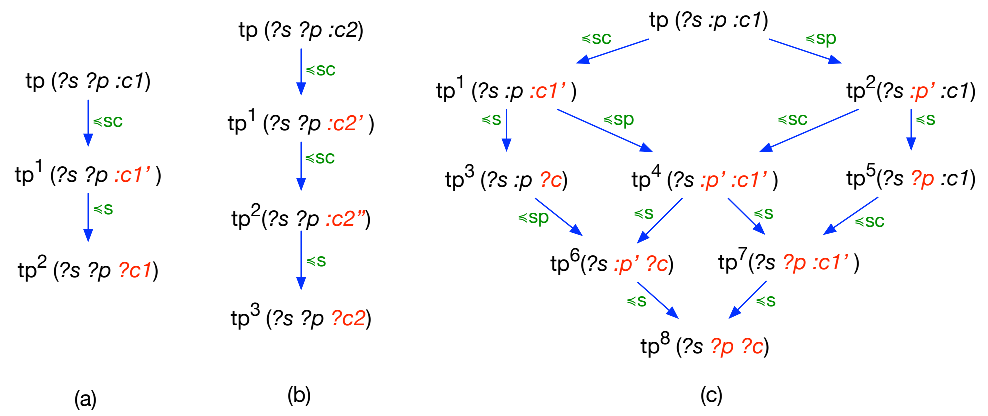


Fig. 3. Relaxation lattices of triple pattern *tp*. ≺sc is a class relaxation, ≺sp is a property relaxation and ≺s is a simple relaxation. In (a), type relaxation is applied to *:c1* once (with superclass *:c1'*) then simple relaxation is used (with the variable *?c1*). In (b), type relaxation is applied twice (with superclass *:c2'* then super-superclass *:c2"*) then simple relaxation is applied (with variable *?c2*). In (c), type relaxation is applied to *:c1* (with superclass *:c1'*), property relaxation to *:p* (with the superproperty *:p'*), and simple relaxation to *:c2'* and *:p'* (with variables *?c* and *?p*.)

3.2.2. Query relaxation

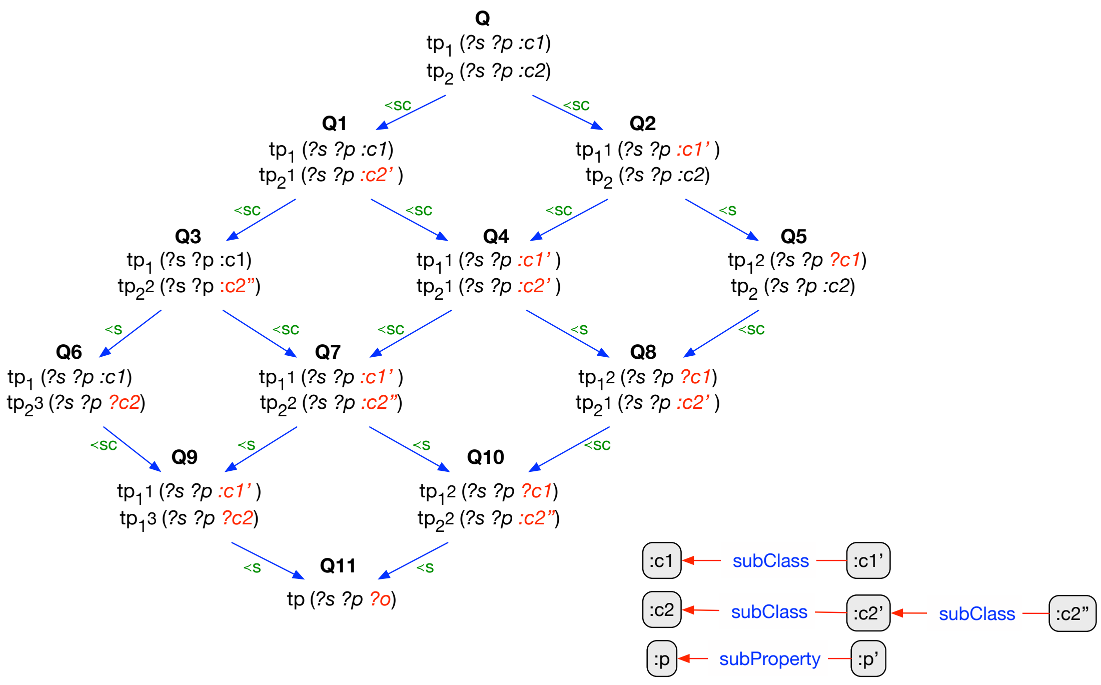
Let Q = X ← P be a (conjuctive) query, where X = (x_1, ..., x_n) is a tuple of variables and P is a graph pattern composed of triple patterns. X is called the head and P is the body. Given two queries Q and Q', Q' is a query logically derived from Q, denoted $Q \prec_Q Q'$, if one or more relaxation 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 $\{Q' = tp_1^1 \wedge \cdots \wedge tp_n^m \mid \exists c \in [1, m] : c > 0\}$ where c counts the number of relaxations of each tp.

Figure 4 shows an example of a relaxation lattice of a query Q. The basic graph pattern of Q is composed of the triple patterns of Figures 3a and 3b. 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*.

3.2.3. Information content measures

Analysing all relaxed queries existing in the relaxation lattice of a query is time-consuming and unnecessary. Information content [3], 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 [21], the information content of concept c can be quantified as the *negative log-likelihood* $-logPr(c)$, 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) ⩽ Pr(c_2). E.g., *Teacher* is-a *Person* then Pr(*Teacher*) ⩽ Pr(*Person*).



Class and property hierarchy

Fig. 4. Relaxation lattice of a query Q that is composed of the triple patterns of Figures 3a and 3b.

Based on information content, [10, 14] 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.

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. Suppose that c' is the superclass of c and p' is the superproperty of p, then the similarity measures between terms can be defined as follows.

- *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 none of the instances belong 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 none of the triples belong to p.
- *Similarity between constants and variables* is $Sim(cst, ?v) = 0$.
  For example, if the object of a triple pattern is a constant and is relaxed to a variable $?v$ using simple relaxation, the similarity in this case is 0.

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 calculated as the average of the similarities between the terms of the triple patterns. The average is usually used to normalize the similarity scores and make them comparable across different terms of the triple pattern.

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

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

Similarity between queries Given two queries Q and Q', such that $Q \prec_Q Q'$, the similarity of the original query Q' to the original query Q, denoted $Sim(Q, Q')$, is the weighted 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 is 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 $Q' \prec_Q Q''$ then $Sim(Q, Q') \geqslant Sim(Q, Q'')$.

3.2.4. 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, or a normalization or scaling process could also be applied to measure the relative position of one numeric value within a predefined range. 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.

3.3. Reification models

Throughout the development of RDF, a series contributions have reshaped the landscape of metadata representation. Some approaches have been proposed to represent specific metadata such as RDFt [22] and tRDF [23] for the temporal annotations, or uncertain data using possibility theory [24]. But more general approaches allow to specify whatever kind of metadata such as standard reification [4], singleton property [6], n-ary relations [25], named graphs [5], RDF-star [7], RDF^M [26], and aRDF [27].

Describing and querying reified RDF triples can be complex and the impact over current relaxation works can be important. In this work, we focus on statement-level metadata, i.e., reification at the granularity of an RDF triple. In the next sections, we analyse five representative reification models, namely, standard reification, named graphs, n-ary relations, singleton properties, and RDF-star. Each approach has advantages and disadvantages. They differ in the number of necessary RDF triples, the use of particular classes or properties, the query syntax, etc. These differences have an impact on the storage volume, the query execution time, the bulk load time, and the integration to the SPARQL specification.

To highlight the main differences among the analyzed reification models, we use a running example where the RDF dataset comprises students, teachers, and courses. Students are enrolled in courses, courses are taught by teachers, and individuals may know each other. In this example, the annotated fact is the enrollment of a student into a course. We consider two annotations: the enrollment date and the type of enrollment payment (Cash or Credit Card). The query of our running example searches for students enrolled in the Semantic Web course before the 13th of September 2023 and who paid by cash.

3.3.1. Standard reification

The standard reification method, proposed in the RDF primer standardized by W3C [4][14] [15], relies on the utilization of IRIs or blank nodes to represent each RDF statement that will be reified. Every statement is defined by three parts according to their roles in the reified statement (rdf:subject, rdf:predicate, and rdf:object). An identifier is assigned to each distinct RDF statement. This identifier is used as subject in the triple that represents the annotation. Figure 5a illustrates the graph representation of a reified statement in our example. A node, identified by the label s100, represents the reified statement (ex:John ex:enrolledIn ex:Semantic_Web). The two annotations (ex:enrolldate and ex:enrollpayment) use this node in the subject. The number of triples for standard reification is 4+n, where n represents the number of annotations. The RDF triples corresponding to this example, are presented in Listing 1a.

Listing 2a shows our example query with standard reification. This SPARQL query contains also four triple patterns that describe the RDF statement and two triple patterns that specify the metadata values using their related predicates. Queries of this reification model are typical subject star-shaped queries.

3.3.2. Named graphs

Carrol et al. [5] extended the RDF data model to cover RDF graphs nameable by URIs. Named graphs were proposed to describe graphs with provenance information[16]. This approach considers pairs of the form (n, g), where n is an IRI that identifies the RDF graph g (n can be a blank node in some cases, or even omitted for a default graph). The graph identifier is used for the annotations. Figure 5b illustrates the idea of named graphs. One RDF graph is defined for the statement(s) to be annotated (g-100), and the annotations are defined into another RDF graph (g-101). This approach is very compact, 1+n triples are needed for n annotations. Listing 1b shows the syntax used in named graphs. Listing 2b shows the query of our running example using named graphs (TriG syntax). Two graphs need to be accessed, the one that contains the statement and the second with the annotations. Each BGP contains is a subject star-shaped set of triple patterns.

3.3.3. N-ary relations

Traditionnaly, a triple links two resources through a relation or property. The n-ary relation model [28][17] was proposed for relations that link a resource to more than one resource or value. This approach proposes to create a new class and n new properties to represent an n-ary relation. An instance of the relation is then an instance of this class. Ontologically the classes created in this way are called reified relations. The instance of a relation can be an IRI or a blank node. Figure 5c illustrates the n-ary relations model where a new class, named ex:Enroll_relation, is created and an instance of this class (ex:enroll_relation_1) is used as a reference for the reified statement. Additionally, ex:Semantic_Web is linked to this instance using the property ex:enroll_value. The same instance is then used for the annotations. This model results in 4+n triples as shown in Listing 1c, n being the number of annotations. Listing 2c shows our example query in n-ary relations. Queries with the n-ary relations approach are composite queries with a subject star-shaped set where the variable searches for the n-ary relation specified in the other part of the query (cf. Line 2).

3.3.4. Singleton properties

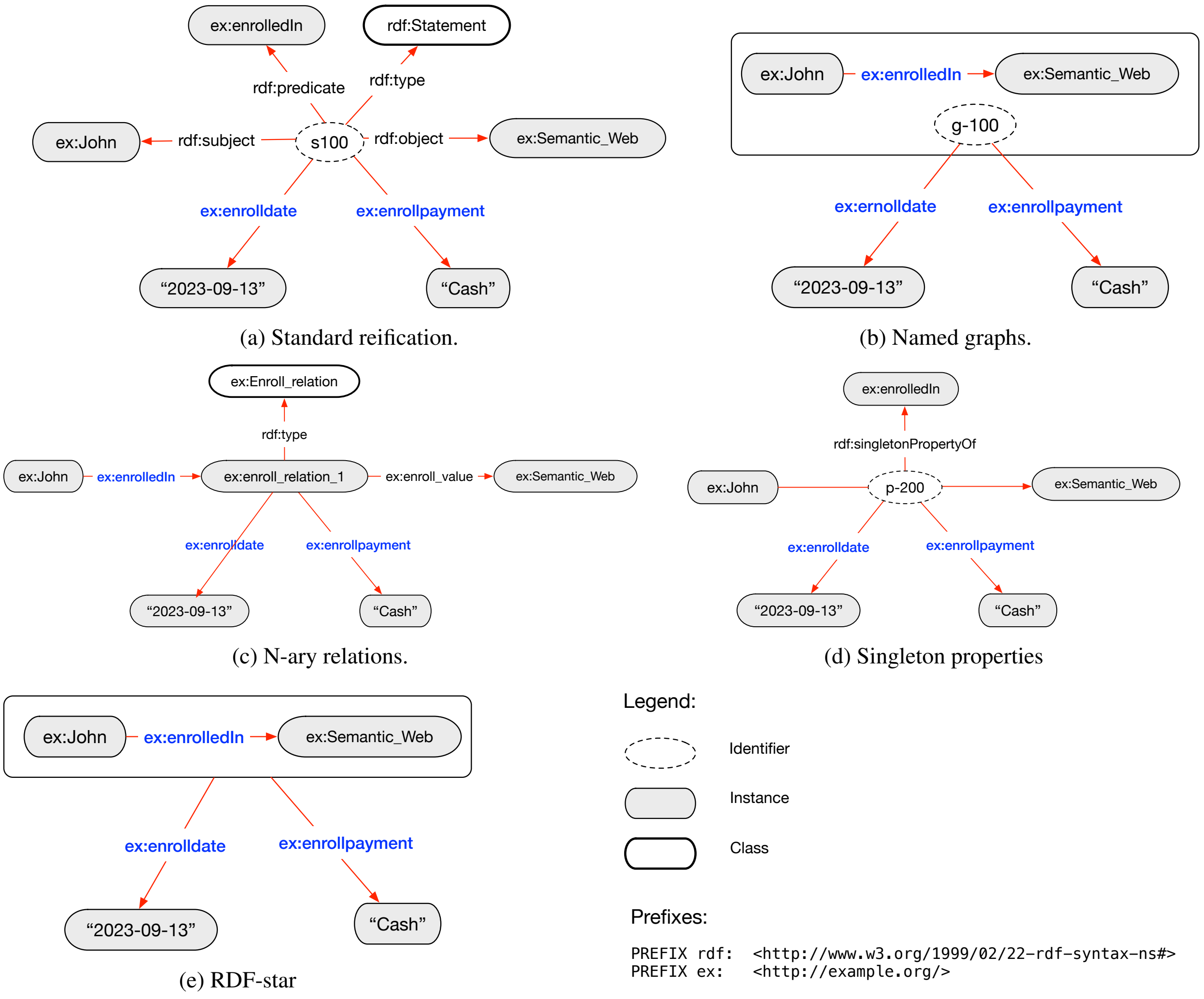
Instead of using new classes, the singleton properties approach [6] proposes to create a unique property for associating metadata. The new property is defined with the property *singletonPropertyOf*. Figure 5d shows the graph representation of this model for our example. In this representation, a new node is created to represent the new property (p-200). This property is linked to the original property via *singletonPropertyOf*. The same property is then used as subject for all annotations. This approach needs 2+n RDF triples to represent n annotations. Listing 1d shows our running example in RDF triples. An example of SPARQL query using singleton properties is shown in

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

[15] https://www.w3.org/TR/rdf11-schema/#ch_reificationvocab

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

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



(a) Standard reification.

(b) Named graphs.

(c) N-ary relations.

(d) Singleton properties

(e) RDF-star

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex:   <http://example.org/>
```

Fig. 5. Illustration of reification models with graph representations.

Listing 2d. A typical query using the singleton approach is a composite query. The particularity is that the chain part is with a predicate variable (cf. Line 2) that searches for instances (same subject variable in Line 3) of the property defined in the star-shaped part of the query.

3.3.5. RDF-star

Recently, [7][18] proposed RDF-star and SPARQL-star, as an extension of RDF and SPARQL. The goal is to provide a compact alternative to standard RDF reification. RDF-star introduces quoted triple, which is a new kind of RDF term. A quoted triple is a triple used as the subject or object of another triple. A quoted triple is written using delimiters ≪ and ≫. The definition of quoted triple is recursive. That is, a quoted triple can itself have a subject or object component which is another quoted triple. Quoted triples can be asserted or not. If they are asserted then they exist as normal (asserted) triples. Figure 5e shows the graph representation of RDF-star with our running example. The quoted triple is inside the rectangular box which is directly linked to the annotations. Listing 1e shows the Turtle-star syntax. In this listing, the quoted triple is not asserted, i.e., it does not appear as a triple.

[18] https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html

```
ex:s100  rdf:type rdf:Statement ;
          rdf:subject  ex:John;
          rdf:predicate ex:enrolledIn;
          rdf:object ex:Semantic_Web ;
          ex:enrolldate "2023-09-13"^^xsd:date;       #1st annotation
          ex:enrollpayment "Cash".                     #2nd annotation
```

(a) Standard reification in Turtle syntax.

```
<g-100> { ex:John ex:enrolledIn ex:Semantic_Web . }
<g-101> { <g-100> ex:enrolldate "2023-09-13"^^xsd:date;       #1st annotation
                  ex:enrollpayment "Cash".                     #2nd annotation
        }
```

(b) Named graphs in TriG syntax.

```
ex:Enroll_relation     rdf:type owl:Class .
ex:John ex:enrolledIn  ex:enroll_relation_1 .
ex:enroll_relation_1   rdf:type ex:Enroll_relation ;
                       ex:enroll_value ex:Semantic_Web ;
                       ex:enrolldate "2023-09-13"^^xsd:date;       #1st annotation
                       ex:enrollpayment "Cash".                     #2nd annotation
```

(c) N-ary relations in Turtle syntax.

```
<p-200>   rdf:singletonPropertyOf ex:enrolledIn .
ex:John   <p-200> ex:Semantic_Web .
          ex:enrolldate "2023-09-13"^^xsd:date;                  #1st annotation
          ex:enrollpayment "Cash".                                #2nd annotation
```

(d) Singleton properties in Turtle syntax.

```
<< ex:John ex:enrolledIn ex:Semantic_Web >> ex:enrolldate "2023-09-13"^^xsd:date;       #1st annotation
                                            ex:enrollpayment "Cash".                     #2nd annotation
```

(e) RDF-star in Turtle-star syntax.

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

SPARQL-star allows to bind a triple pattern enclosed in angle brackets as subject or object on other triple pattern. Our query example is shown in Listing 2e. This syntax shows that with this nesting-based approach, only one triple pattern is needed to search in a reified statement. It is hard to classify the shape of SPARQL-star queries. With our example query we can identify a subject star-shaped triple pattern in the enclosed angle brackets which is used for searching matchings with the given annotations.

RDF-star and SPARQL-star are still not a W3C recommendation. However, RDF 1.2[19] will introduce quoted triples.

3.3.6. Comparison of reification models

The described reification models differ in various criteria such as the number of triples (resp. triple patterns), flexibility, syntax support, and human understandability, etc.

[19] https://www.w3.org/TR/rdf12-concepts/

```
SELECT ?student WHERE {
  ?statement      rdf:type               rdf:Statement;
                  rdf:subject            ?student;
                  rdf:object             ex:Semantic_Web;
                  rdf:predicate          ex:enrolledIn;
                  ex:enrolldate          ?date.
                  ex:enrollpayment       "Cash".
  FILTER (?date <= xsd:date("2023-09-13"))
}
```

(a) Standard reification.

```
SELECT ?student WHERE {
   GRAPH ?g {
               ?student ex:enrolledIn  ex:Semantic_Web .
            }
   GRAPH ?m {
               ?g  ex:enrolldate ?date;
                   ex:enrollpayment "Cash".
             }
   FILTER (?date <= xsd:date("2023-09-13"))
  }
```

(b) Named graphs.

```
SELECT ?student WHERE {
     ?student     ex:enrolledIn    ?t .
     ?t           rdf:type ex:Enroll_relation ;
                  ex:enroll_value  ex:Semantic_Web ;
                  ex:enrolldate ?date;
                  ex:enrollpayment "Cash".
    FILTER (?date <= xsd:date("2023-09-13"))
 }
```

(c) N-ary relations.

```
 SELECT ?student WHERE {
     ?er      ?p  ex:Semantic_Web.
     ?p       rdf:singletonPropertyOf  ex:enrolledIn;
              ex:enrolldate ?date;
              ex:enrollpayment "Cash".
    FILTER (?date <= xsd:date("2023-09-13"))
  }
```

(d) Singleton properties.

```
 SELECT ?student WHERE {
      << ?student ex:enrolledIn ex:Semantic_Web >> ex:enrolldate ?date;
                                                   ex:enrollpayment "Cash".
      FILTER (?date <= xsd:date("2023-09-13"))
     }
```

(e) RDF-star.

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

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.[20]

Number of triples. Standard reification is the most costly approach since it needs 4+n triples for n reified statements. N-ary relations needs 3+n triples plus a new class declaration. This class declaration is done once by statement whatever the number of annotations. The singleton model needs 2+n triples. Named graphs and RDF-star are the most compact models needing 1+n and n 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 named graphs model allows these different granularities for annotations.

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 several objects sharing the same subject-predicate pair), 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. The number of necessary triple patterns follows the ranking of the number of triples. Standard reification is the model that needs the most number of triple patterns in a SPARQL query. 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.

Query shapes. The most common query shapes used to query metadata are the star and the composite shapes. N-ary relations and singleton properties use composite queries with a subject star-shaped part. Notice that the chain part of singleton properties queries is over a predicate (predicate and subject instead of the traditional object and subject). Standard reification uses typical subject star-shaped queries. Named graphs uses star-shaped queries but accessing potentially several graphs (two graphs in our example). SPARQL-star queries are nested queries that in our example contain a star-shaped triple pattern.

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 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 statements and their annotations. It does not involve refactoring the statement, declaring new predicates or new classes. This facilitates human understandability.

Several works studied different reification methods and compared them according to several criteria. To mention a few, [25] focused on Wikidata and its representation in RDF using reification based on n-ary relations, standard reification, singleton properties and named graphs. [29] realized an analysis of standard reification, named graphs, n-ary relations, singleton properties, companion properties (proposed in that paper) and RDF-star in its early stages. [30] experimentally analysed the behaviour of standard reification, singleton properties, named graphs, and RDF-star over a knowledge graph with a huge number of annotations (over 70% of the used RDF dataset). [31] compared standard reification, n-ary relations, RDF-star and Qualifiers [25] (the reification model proposed for Wikidata) for three consumer scenarios: knowledge exploration, systematic querying, and embeddings for graph completion. [32] proposed REF, 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 a single triplestore using three reification models (standard reification, singleton properties and RDF-star). [33] proposed StarBench (as an extension to REF), a benchmark for testing the SPARQL-star support and runtime performance. Their analysis highlighted limitations in loading the data, query parsing, correct evaluation of SPARQL-star queries, as well as contrasting query execution performance across all tested engines.

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

```
PREFIX ex:  <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?student WHERE {

    ?teacher     rdf:type                  ex:Teacher;                  #tp1   Querying data
                 ex:teaches                ex:Semantic_Web.             #tp2
    ?student     rdf:type                  ex:Student;                  #tp3
                 ex:friendOf/foaf:knows    ex:Alice;                    #tp4
                 foaf:knows                ?teacher.                    #tp5
    ?statement   rdf:type                  rdf:Statement;               #tp6   Querying metadata
                 rdf:subject               ?student;                    #tp7
                 rdf:object                ex:Semantic_Web;             #tp8
                 rdf:predicate             ex:enrolledIn;               #tp9
                 ex:enrolldate             "2023-09-13"^^xsd:date.      #tp10
         }
```

Fig. 6. SPARQL query Q used in our running example.

Results suggest that there is no outright winner for all comparison aspects. Standard reification and named graphs appear to have good performance. Singleton properties frequently produces poor performances. This is likely due to the high number of unique properties, and indexes of triplestores are usually not optimized with that in mind. Finally, it appear that RDF-star should be more efficiently implemented, as it is becoming part of the RDF 1.2 specification.

The next section surveys SPARQL query relaxation approaches. Even though they have proven their efficiency, RDF reification poses significant challenges and raises open questions. Is it pertinent to relax terms introduced to employ various methods of statement-based reification? Is it crucial to distinguish the part of a query that searches for annotations from the one that searches for data? Can annotation values be relaxed similarly to traditional description values? Does the higher number of triples and triple patterns necessary to describe annotations have a performance impact on query relaxation approaches?

4. Review of SPARQL query relaxation approaches

We organise the survey of existing relaxation works in two sections. Section 4.1 analyses approaches focusing on relaxation using RDFS ontologies and entailment rules [8–12, 14, 16, 18–20]. Then, Section 4.2 surveys works oriented to relaxation based on similarity of instances [13, 15, 17]. For each work we describe its approach, the experimental setup and evaluation conclusions. Table 3 shows analysed relaxation approaches with a brief description of their main contributions.

The SPARQL query of Listing 6 will be used as running example as much as possible. This query searches for students enrolled in the Semantic Web course who are friends of Alice and know the Semantic Web teacher. This query searches for data (triples) but also for metadata (triples with RDF reification). We use the standard reification model to faciliate our explanations. Figure 7 introduces the ontology we use in our example. Appendix A shows supplemental material about our running example. A.1 lists the example dataset. A.2 lists the used ontology in Turtle. A.3 lists the dataset closure. A.4 shows the statistics of classes (Table 8), properties (Table 9), and some query similarity scores (Table 10). The size of the query relaxation lattice is approximately two million relaxed queries. A.5 shows the relaxed queries of the first level of the query relaxation lattice composed of seventeen queries. Notice that only Q_{12} and Q_{17} return results.

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.

Reference	Contribution		Description
[8, 9, 11]	Extends the OPTIONAL clause with a RELAX clause.		Extends the OPTIONAL clause using the RDFS entailment rules and the *extended reduction of the ontology*.
[12]	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 built from the properties describing entities.
[10, 14]	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.
[16]	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.
[18]	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.
[19]	Query relaxation using SHACL constraints		Proposes an ontology-based relaxation streategy with focus on the domain and range restrictions and property paths, ultimately improving the retrieval and ranking of relevant information.
[20]	License-aware query relaxation (FLiQue)		Proposes a license-aware query relaxation strategy that supports federated queries and prevents license conflicts.
[13]	Statistical language model relaxation.		Uses NLP techniques (statistical language models) and the Jensen-Shannon divergence to relax entities and predicates.
[15]	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.
[17]	Partitioning-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.

4.1.1. *Extension of the OPTIONAL clause with a RELAX clause*

Hurtado et al. [8, 9, 11] propose a RELAX clause as a generalization of the OPTIONAL clause of SPARQL for conjunctive queries. These works use four types of relaxation: type relaxation, property relaxation, typing relaxation (using the domain and range rules), and simple relaxation. Simple relaxation includes dropping triple patterns (similar to the OPTIONAL clause of SPARQL), replacing constants with variables, and breaking join dependencies. Breaking join dependencies consists in generating new variable names for a variable that appears in multiple triple patterns. 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. But the queries that are on the same relaxation level are not distinguished.

It is worth noting that the approach introduced in [8] focused on single triple pattern queries or simple chain queries. [9] extends their relaxation approach by addressing graph patterns (using the product of the relaxation lattices of the graph pattern). Then, [11] 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. For instance, the property path ex:friendOf/ex:knows of our running example will produce two triple patters, and ex:friendOf will be relaxed to ex:knows. This will interestingly result

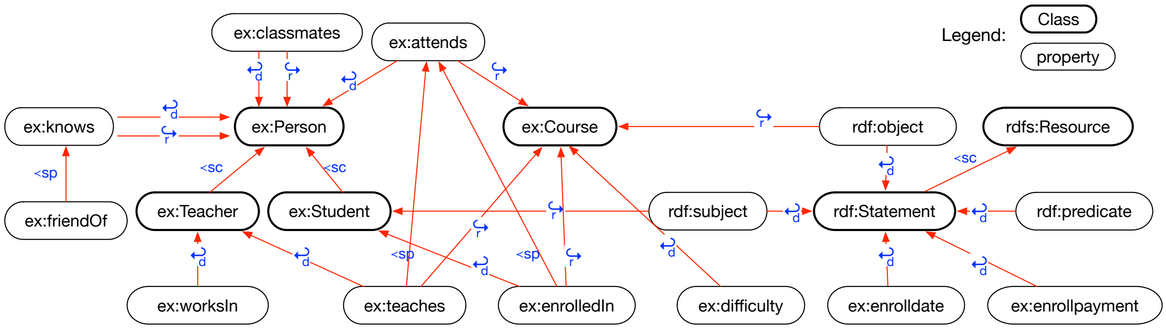


Fig. 7. Ontology of our running example.

in obtaining extra answers. In our example dataset, this relaxation will return the student Maya who knows Peter, who in turn, knows Alice.

The RELAX clause proposed in these works has interesting constraints. A triple pattern whose predicate term is in the RDFS vocabulary cannot be relaxed except for type-triple-patterns (i.e., triple patterns having rdf:type in the predicate). This kind of triple pattern will be relaxed only with Rule 4 (rdfs9) of Table 2, where a class is replaced by a superclass in the object. Applied to our running example, the metadata part of our example query would not be relaxed except for the triple pattern $tp6$. Thus, $tp6$ can be specified as *RELAX{?statement rdf:type rdf:Statement}*. In that case, the rdf:Statement class will be relaxed with its superclass rdf:Ressource. This relaxation is semantically useless because numerous things are instances of rdf:Resource (all entities in subjects and objects). See Q_9 in the Appendix A.5.

We remark that the works by Hurtado et al. propose theoretical concepts and algorithms but do not include experimental evaluations of their contributions.

4.1.2. Ranking model based on Bayesian networks

Huang et al. [12] propose an approach to rank the relaxed queries according to their similarities to the original user query, and an original method to compute similarity values using Bayesian networks. The goal is to identify a set of relaxed queries to obtain k-relevant answers. This work considers type relaxation, property relaxation, typing relaxation and simple relaxation. It focuses on relaxing predicates and objects within the context of subject star-shaped queries to find similar entities.

The similarity of a relaxed query to the original query is measured based on the difference of their estimated results, i.e., their selectivity, or more concretely $Pr(c)$ (see Section 3.2.3). The main contribution lies in estimating selectivity by considering the correlation among properties that describe 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 uses Bayesian networks as an appropriate structure to divide the joint probability distribution into several distributions of fewer variables by using the conditional independence assumption. The structure of these Bayesian networks is learned using the K2 algorithm [34]. 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 offline, previous to the online query processing.

This approach allows to discard relaxed queries with empty results. Our first observation is that as this approach focuses on star-shaped queries, the data part of our example query will be analyzed separately from the (related) metadata part (reification). Triple-patterns like $tp6$ can be relaxed leading to uninteresting results. The advantage is that the ranking strategy will calculate a low similarity when this triple pattern relaxation is considered. A limitation of the ranking strategy is that it does not distinguish between queries that return the same number of results. For instance, Q_{12} and Q_{17} of our running example will not be distinguished because they return the same number of answers (see Appendix A.5).

Experimental setup. The experiments conducted use Jena[21] and the Lehigh University Benchmark (LUBM) [35]. LUBM relies on a university domain ontology with numerous properties and classes. The experiments involve five star queries, each comprising three or four triple patterns. To compute k-relevant approximate answers, an algorithm based on best-first search is implemented, with the parameter k set to specific values.

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

[10, 14] proposed the query similarity measures introduced in Section 3.2.3. [14] proposes and evaluates three query 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 query relaxation lattice and execute the relaxed queries in a ranking order according to similarity measures to generate the k-relevant answers for a query.

- BFSR selects the best-relaxed query (based on the similarity measures) from the relaxed queries at every level of the query relaxation lattice until enough answers are obtained.
- OBFSR is an optimization of BFSR that checks the usefulness of relaxed queries and skips unnecessary relaxed queries. It considers the join dependency between the triple patterns and the data distribution in the dataset to prune the relaxation lattice. OBFSR will prune the lattice by avoiding the evaluation of relaxed queries in cases where classes and properties are relaxed to their superclasses and superproperties that do not have new instances. Coming back to our running example, a relaxed query with the class ex:Person instead of ex:Teacher in $tp1$ will produce no new instances because $tp2$ has in its domain the class ex:Teacher. So, OBFSR will avoid evaluating relaxed queries that join these two triple patterns (see Q_1 in the Appendix A.5).
- BR studies the problem of relaxing queries as a batch-based process to accelerate the query processing. The proposed idea is to execute relaxed queries as a group (batch) and skip some unnecessary relaxed queries for obtaining k-relevant answers. It improves OBFSR by considering the cases where the answers of a relaxed query are included in the answers of another query. The batch size is the number of relaxed queries that need to be executed to obtain k-relevant answers. BR excels when users require a large number of approximate answers, making it ideal for scenarios emphasizing result quantity over precision. In our running example, if $tp2$ is relaxed (ex:teaches to ex:attends), and Q_4 relaxes the object of tp_2 with simple relaxation. If both queries are part of the batch, as $Q_3 \prec_Q Q_4$ and so, answers of $Q_3 \subseteq Q_4$ then executing Q_3 is not necessary.

Although these algorithms rank relaxed queries based on similarity scores, distinguishing between relaxed queries can still be challenging. Consider two relaxed queries generated by simple relaxation. Despite both queries may yield distinct answers, it would be difficult to distiguish them if they have same similarity measures (see last line of Table 10 in the Appendix A.4).

Experimental setup. These algorithms were evaluated using Jena TDB[22]. [14] constructed a dataset using LUBM. 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.

Evaluation. Experiments showed that when the user asks for few answers, BR will behave as OBFSR and BFSR. OBFSR is more efficient than BFSR since it prunes the lattice and saves more time. 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. The experiments carried out do not present a comparison with other state-of-the-art contributions.

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

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

4.1.4. Query relaxation strategies based on failure causes

Fokou et al. [16] present three approaches that focus on optimizing the relaxation process by investigating the failure causes of relaxed queries. Authors argue that executing relaxed queries in a similarity-based ranking order lacks of efficiency because queries are relaxed without identifying their failure causes. The failing sets of triple patterns in a query are called Minimal Failing Subqueries (MFS). Identifying the MFSs (i.e., the set of MFS) of a query is an NP-hard problem. In [18, 36], authors propose efficient methods to computate the MFSs of a failing query. Discovering 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. [16] 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 rank relaxed queries based on the similarity function described in Section 3.2.3. 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., MFSs. 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. When applying MBS to our running example, the only MFS consists of $\{tp_4 \,.\, tp_7 \,.\, tp_8 \,.\, tp_{10}\}$. This means that there exist no student in our dataset who is enrolled in the semantic web course on 2023-09-13 and who has a friend who knows Alice. This allows to skip all the relaxed queries that contain this MFS.
- 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 at least one MFS has not been relaxed in Q'. However, there could exist repaired queries that still fail. Queries with relaxed MFS are executed and if they give no result they are pruned as well as all those relaxed queries having the same repaired MFS. By applying O-MBS on the example query, a repaired query with $tp4$ relaxed with simple relaxation on the object (ex:Alice is replaced with a variable), still fails to return an answer (see Q_7 in the Appendix A.5). Thus, O-MBS will detect it and skip it as well as all relaxed queries having the same repaired subquery.
- F-MBS extends the O-MBS strategy by discovering all failing causes of every relaxed query. 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. This strategy will skip all queries in the Appendix A.5 except Q_{12} and Q_{17} and will go forward in the relaxation lattice.

Experimental setup. These three algorithms were implemented in Java using Jena TDB and Virtuoso[23]. Datasets were generated with LUBM. Seven queries are used in the evaluation experiments, where these queries include the main query patterns (star, chain, and composite).

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. However, these experiments do not show a comparison between these proposed algorithms and other state-of-the-art contributions.

4.1.5. Relaxing failing queries based on the hitting set problem

Mebrek et al. [18] introduced an approach named *CADER* for handling failing queries. Unlike all previous proposed relaxation techniques, CADER relaxes the constraints of a query only by eliminating triple patterns. It does not use any ontology-based relaxation or query similarity. However, this approach also aims to optimize the relaxation process by studying the failure causes of the relaxed queries similar to the works described in Section 4.1.4 but in different manner. It involves two main steps: (1) determining the MFSs, and (2) exploring these failing subqueries to compute the set of maximal succeeding subqueries (XSSs). The basic idea of identifying MFSs consists in decomposing the initial query into the smallest subqueries (one triple pattern) and evaluating those queries until finding

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

the largest queries with empty answers. In [16], the idea is to start with queries having the largest number of triple patterns and progressively move to those with the smallest number. The iterative browsing performed by CADER is more efficient since it reduces the exponential search space, resulting in the evaluation of fewer queries. If a query fails to give answers, then all its proper superqueries will fail. Consequently, evaluating these superqueries becomes unnecessary. Regarding the second step, this approach models the query relaxation problem as a hitting set problem. Given a collection of subsets (i.e., answers calculated in the step 1), the hitting set problem is finding the smallest subset that intersects (hits) every set in the collection. This is performed by discovering the MFSs and searching for their hitting sets which correspond to the complement set of XSSs. 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 (the MFSs).

By applying CADER on the motivating example, it will identify the MFS of the original failing subquery that was determined in Section 4.1.4 (i.e., $\{tp_4 \ . \ tp_7 \ . \ tp_8 \ . \ tp_{10}\}$). Then it will discover the set of XSS which is a set of subqueries excluding the triple patterns that are in the discovered MFS. So, the set of relaxed queries are the 4 subqueries that exclude the triple patterns t_4, t_7, t_8, and t_{10}.

Experimental setup. CADER has been implemented in Java with the Jena library. The experiments are carried out using datasets from DBpedia and LUBM. Several experiments were performed to compare the proposed approach with the LBA and MBA algorithms proposed in [36]. Queries of varying types (star, chain, and composite) were generated over the DBpedia and LUBM dataset. The LUBM tested queries were borrowed from [36].

Evaluation CADER behaves the best regardless the types of queries in terms of execution time.

4.1.6. *Query relaxation using property paths and SHACL constraints*

Lahmam Bennani et al. [19] propose two original predicate relaxations. The first one, relaxes a predicate to a predicate+, where + is a property path operator. The second one relaxes a predicate with another predicate based on the same domain and range constraints. The idea is to relax a property with another whose domain or range includes the subject or object in the triple pattern. To generate this relaxation, the SHACL constraints language [37] is used because it provides more expressive domain and range constraints than RDFS. Similarities are computed like in Section 3.2.3. This work explicitly uses weights to give priority to some triple pattern relaxations. However, the similarity is set to 0 when relaxing predicates using domain and range restrictions and to 1 when relaxing to a property path. This gives property path relaxations more preference. Simple relaxation is not used in this work. Only predicates and objects are relaxed. Suppose that there exist two constraints that say that only a student can enroll in a course and only a teacher can enroll in an employee meeting. In that case, SHACL shapes can be defined for identifying these constraints and defining the allowed domain and range of the predicate related to enroll. This approach will work on relaxing the predicate while still satisfying these constraints.

In addition, predicates will be transformed into property paths (for example, ex:teaches+, ex:knows+, etc.).

Experimental setup. This work was evaluated over a dataset named Brick which its ontology is composed of building representations. A set of basic conjunctive queries were defined for this dataset. In the experiments, two scenarios are considered. In the first scenario, all the weight values for computing the similarity measure are 1, while in the second scenario, there is a willingness to relax a set of triple patterns in the original query. As a result, lower weights are assigned to some triple patterns to prioritize relaxation over these specific patterns.

Evaluation. Experimental results show that a relaxation time less than 1 ms was observed for most of the queries. However, these experiments do not present a comparison with other state-of-the-art contributions. Furthermore, the results of the two considered scenarios for weight assignments demonstrated that assigning weights could facilitate the retrieval of more answers at lower levels of the relaxation graph.

4.1.7. *License-aware query relaxation*

Moreau et al. [20] proposes FLiQue, a license-aware query processing strategy that uses query relaxation to deal with contradictions among licenses. This is the only relaxation approach among the proposed works that deals with license conflicts in a federation of SPARQL endpoints. 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. 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 technique (O-MBS proposed in Section 4.1.4). 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. 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 [38], a lattice-based model for license orderings. 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 is used to produce and propose relaxed queries. This work does not propose a new query ranking strategy (queries are ordered by similarity like in O-MBS), but it is able to propose queries ordered by their licenses, from the less to the most restrictive license.

The FLiQue approach can be applied to our example and its behaviour would be similar as for O-MBS. The difference is that FLiQue can be also used in a federation of endpoints and can deal with the exponetial number of possibilites wich depend on the instances of each dataset.

Experimental setup. FLiQue was implemented in Java using Jena and CostFed [39], 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 [40], a benchmark for a federation of SPARQL endpoints. This benchmark contains 32 federated queries (14 simple queries, 10 complex queries, and 8 large queries). Queries were executed over a federation of 11 interlinked endpoints.

Evaluation. The experiments aim to demonstrate FLiQue's ability 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. 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 depth of the hierarchies of the ontologies concerned by LargeRDFBench datasets.

4.2. Relaxations based on similarity of instances

Next 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. [13] see the query relaxation problem as the entity-relationship-oriented counterpart of query expansion in traditional keyword-search settings. It uses NLP techniques, particularly statistical language models, to relax queries. 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 combining unigrams and bigrams using linear interpolation smoothing from the corpus (i.e., the knowledge graph). The language model of each entity or relation becomes 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 is conducted offline prior to 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. External information sources could be also used in this work to relax entities and predicates.

In our query example, the course ex:Semantic_Web can be replaced with the course ex:Databases by estimating their statistical language models. This estimation can be done from some common information about these two courses in the dataset, such as that both are taught by ex:Patricia. Concerning RDF reification, reified triples may induce errors because, for instance, in standard reification the subject is an identifier not an entity. But, it is possible that in a query relaxation, the subject of triple patterns like tp_6 be replaced by another identifier with similar reification.

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).

Evaluation. Experiments based on human evaluations were conducted to evaluate the quality of the relaxations and relaxed query results. Evaluators studied the quality of these relaxations and how close they were. The same was carried out for the relations. 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. This work does not report a comparision with other approaches.

4.2.2. Relaxations of heterogenous resources descriptions

Hogan et al. [15] focus on entity-lookup queries using similarities of RDF terms. They address vague queries executed over heterogenous and incomplete data. Vague means that the query issuer may be not sure about the query constraints. When queries yield no results, this approach proposes rewriting them with similar constraints. This work introduces a generic framework to calculate distance scores between resources based on their structured descriptions. This process is conducted offline before online query processing. The target queries are subject-star shaped queries where only objects are relaxed. The framework maps entity values to relaxation scores using multiple distance functions. These scores range from 0 to 1, where 1 indicates no interchangeability, and 0 indicates perfect interchangeability. For instance, distance(:blue, :navy) might be 0.2, suggesting :navy as a good relaxation for generic :blue. The choice of the distance functions depends on attribute types, such as normalized distances for numeric attributes or Levenshtein distance for string attributes. Background knowledge, including similarity tables, can also be used. This approach positions the original query as the reference point and maps entities into an n-dimensional space. The closest entity value to the query is chosen. An overall relaxation score is computed for each entity based on its distance from the original query, facilitating result ranking. Users can assign vagueness scores to attribute-value pairs to control relaxation by property. It also proposes a concurrence measure that matches RDF resources based on the cardinality of property-value pairs in the dataset.

Considering our running example, a query that searches students enrolled in a “very easy” course, will give no answer as in our dataset there are no courses described as very easy (only easy and hard). But, if we use the Levenshtein distance, “easy” can replace “very easy” because “easy” is closer to “very easy” than “hard”.

Experimental setup. The proposed relaxation framework was tested against the vehicles dataset of the EADS project (European Aeronautic Defence and Space Company) which is considered a small dataset. The effectiveness of this framework was also evaluated over three small queries. These queries were defined from their expression in natural language. Then, handmade, the expressions were transformed into SPARQL queries with vagueness scores. Transforming natural language phrases automatically into queries is out of the scope of this paper.

Evaluation. Top 5 approximate answers of each query were collected and evaluated using scores based on root-mean-square deviation. From these observations, it was 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. Therefore, it becomes challenging to compare these values and determine the most suitable relaxation, making it difficult to distinguish the most relevant answers.

4.2.3. Partitioning-based relaxation

Ferré [17] proposes a strategy for finding approximate answers and optimizing the evaluation of relaxed queries. 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 instance-based relaxation but it can also behave like an ontology-based relaxation approach. The foundation of this strategy is Formal Concept Analysis [41], a mathematical theory for deriving implicit relations between objects based on common attributes. In the context of semantic web queries, objects refer to RDF nodes, and attributes are the properties or characteristics associated with these nodes. When we formulate a query in the semantic web, it essentially acts as a description of RDF nodes. The query specifies certain conditions or properties that we are interested in when searching for information in a knowledge graph. Consider two classes in a knowledge graph, such as Course and Talk. These classes may have common properties, like duration, topic, and audience. Instances of these classes are the specific nodes in the graph, and when these instances share similar properties, they can be grouped together as part of a broader concept. For instance, if there are instances of both classes Course and Talk with same duration, topic, and audience, these instances could be identified as part of a broader concept—let's call it EducationalEvent. In this context, the query represents a description of RDF nodes, and relaxed queries symbolically describe similar nodes. This work relies on symbolic similarity rather than numeric measures and utilizes relaxation distance to establish a partial ordering instead of a total ordering.

This paper proposes two algorithms: Answers partitioning and Lazy join.

- Answers partitioning. This algorithm is the relaxation core of this work. It starts from a general query and progressively adds triple patterns to create more specific queries, efficiently eliminating irrelevant answers. This approach contrasts with traditional top-down approaches in query relaxation. The choice of triple patterns is free from the set of triple patterns in the original query.
- Lazy join. The Lazy join algorithm is proposed to optimize the evaluation of the relaxed queries generated by the Answers partitioning algorithm. Unlike the traditional approaches, this algorithm passes reversely from the relaxed queries to the more specific queries. This guarantees less complexity since the newly generated answers are a subset of the previous one. This approach is advantageous for queries with multi-valued properties, as it avoids the explosion in join size.

This approach starts initially with a single partition defined by the fully relaxed query, i.e., the query with an empty body and the set of all possible answers. In addition, a set of candidate triple patterns is identified, containing all the triple patterns from the initial query that failed to produce enough results. Then each partition will be split in two parts by using a triple pattern as discriminating criteria generating a binary tree. As the choice of triple patterns is free, in our running example, the triple pattern (?student rdf:type ex:Student) could be chosen first. Based on this, the initial partition is split into two partitions. The first partition consists of a relaxed query that contains only this triple pattern and a set of answers that validate this query (ex:John, ex:Maria, and ex:Maya). While the second partition will remove this triple pattern and the results will be the complement of the results of the first partition. The algorithm stops when no partition can be split further. This algorithm is anytime because a partition is defined at all time so it can be stopped at any level. The partitions can be split in any sequence, enabling the adoption of either depth-first or breadth-first strategies, or the application of heuristics. The selection of the splitting triple pattern is also unrestricted and remains independent from one partition to another.

Experimental setup. This approach was implemented in OCaml[24][25]. The experimental evaluation involved multiple datasets, including MONDIAL[26], LUBM10, and LUBM100, with varying query sizes. 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.

Evaluation. The experimental results showed the efficiency of these algorithms in query relaxation over the small queries. In addition, it was proved that these algortihms are capable of exploring the search space in the absence of ontological definitions for queries having up to 1500 triple patterns. 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.

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

[25] 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

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

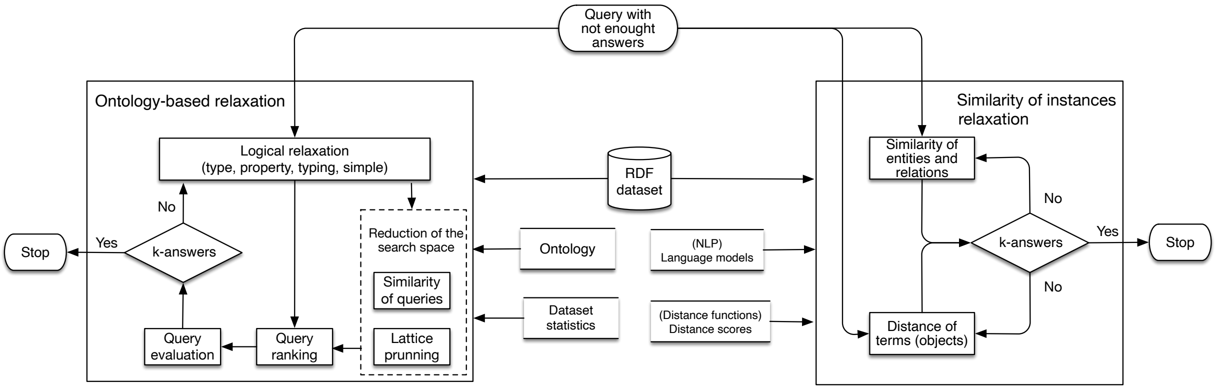


Fig. 8. Global query relaxation process of the two types of analyzed works (ontology-based and similarity of instances).

5. Comparision of relaxation approaches and applicability over RDF reification

The relaxation processes of analyzed works share several common aspects. Figure 8 groups those relevant aspects from both types of analyzed works. The starting point is a query with empty or not enough answers. On the left part of the figure, the ontology-based relaxation process will logically relax the query constraints based on the class and properties hierarchies but also the domain and range descriptions existing in the ontology. Simple relaxation is not based on the ontology but it allows to generalize constraints by replacing terms with variables. As the number of relaxation possibilites is huge (generating all the relaxation possibilitites of the query relaxation lattice is a NP-hard problem), partial orders are calculated to rank queries by relevance. To reduce the search space of the most relevant relaxed queries (cf. doted box in Figure 8), it is possible to calculate the similarity of relaxed queries using the information content (dataset statistics). This allows to identify relevant query rankings so that the query relaxation lattice will be pruned (unrelevant queries are discarded). Then, there is an iteration process where relevant queries are generated and evaluated until reaching k-relevant answers.

On the right part of Figure 8, the general approach of the similarity of instances process is shown. This approach needs a potentially costly offline process such as building probabilistic language models or defining relaxation scores for every pair of entity values using various distance functions. This enables efficient relaxation of entities, relations, or object-terms during query execution. Then during query execution, the relaxation process can be done efficiently to relax entities, relations or object-terms.

If we generally position the SPARQL query relaxation contibutions, Hurtado et al. [8] initially introduced the concept of relaxing a triple pattern by extending the OPTIONAL clause using RDFS entailment rules and ontology. Building upon this foundation, [9, 11] extended the scope to handle not only individual triple patterns but also SPARQL queries and property path queries. Huang et al. [12] contributed by proposing a ranking strategy. Their approach aimed at ordering relaxed queries and distinguishing them based on their similarity to the original query. To address the challenges posed by the search space, [14, 16] made significant improvements by introducing optimizations. These optimizations helped in reducing the search space during query relaxation. [18] stands out in terms of efficiently computing the failing relaxed queries. Moreau et al. [20], added the data distribution aspect at considering a federation of SPARQL endpoints instead of the traditionally single SPARQL endpoint. Some notable works [13, 15] proposed approaches to relax queries in the absence of explicit ontological definitions. Finally, the partitioning-based work [17] introduced optimized algorithms, showcasing efficiency in similarity search for large queries in situations where ontological information is absent.

Analyzed works are organized in two parts, the first one focuses on ontology-based relaxation and the second one on instances-based relaxation. Our first observation is that most of the works are included in the first part (see the first eleven lines of our tables). Only three out of the fourteen analyzed approaches propose instances-based relaxations. Notice that from these three works, [17] can also use ontology-based relaxation. This section compares the fourteen reviewed approaches with an analysis of relevant aspects about query relaxation. We begin by drawing a general comparison in Section 5.1. In Section 5.2 we compare reviewed approaches based on particular aspects

Contribution	**Necessary Information**			**Query Shape**				**Relaxed Terms**		
	RDF dataset	Ontology	Data summaries	Star queries	Chain queries	Composite queries	Other	Subject	Predicate	Object
Extension of the OPTIONAL clause with a RELAX clause [8, 9, 11]	✓	✓	✗	✓	✓	✓	Regular path queries	✓	✓	✓
Ranking model based on Bayesian networks [12]	✓	✓	✗	✓	✗	✗		✗	✓	✓
BFSR, OBFSR, BR [14]	✓	✓	✗	✓	✓	✓		✓	✓	✓
MBS, O-MBS, F-MBS [16]	✓	✓	✓	✓	✓	✓		✓	✓	✓
CADER [18]	✓	✗	✗	✓	✓	✓		✗	✗	✗
Query relaxation for portable brick-based applications [19]	✓	✓	✗	✓	✓	✓		✗	✓	✓
FLiQue [20]	✓	✓	✓	✓	✓	✓		✓	✓	✓
Statistical language model relaxation [13]	✓	✗	✗	✓	✓	✓		✓	✓	✓
Relaxation of heterogeneous resources descriptions [15]	✓	✗	✗	✓	✗	✗		✗	✗	✓
Partitioning-based relaxation [17]	✓	✓✗	✗	✓	✓	✓		✓	✓	✓

Table 4

General comparison of analysed works.

of the query relaxation process. In Section 5.3 we analyze and compare their experimental evaluations. Section 5.4 positions the query relaxation in the context of RDF reification. Our analysis is summarized in comparative tables (Tables 4, 5, and 6) where ✓ means the comparative criteria is satisfied, ✓✗ means that it is more or less satisfied, and ✗ indicates absence of information.

5.1. General comparison

Query relaxation is a costly process because numerous relaxed queries can be evaluated before obtaining the most relevant k answers. The necesary information for the query processing relaxation is the RDF dataset (to evaluate candidate queries), the ontology (to relax queries in ontology-based relaxation), dataset statistics (to optimize the relaxation processing cost). Depending on the kind of relaxation, analyzed approaches use some or all this information.

Concerning the SPARQL query characteristics of relaxed queries, in general only BGPs are relaxed (set of conjunctive triple patterns). These works do not focus on relaxing FILTER, or UNION queries. The OPTIONAL operator allows somehow a relaxation of the query constraints as the OPTIONAL triple patterns are joined only if triples are mapped. Analyzed works do not relax OPTIONAL triple patterns. One work proposes instead an extension of this operator with a RELAX clause.

All works are able to relax star-shaped queries. And only one approach is able to relax property path queries. This observation has a logical explanation because the most used queries in real applications are star-shaped queries. Property path queries are relatively new (since SPARQL 1.1.), and they can be complex to evaluate. Concerning the relaxed terms (subject, predicate, object), all of them can be relaxed.

Next paragraphs detail our general comparison that is summarized in Table 4.

Necessary information. The second column of Table 4, shows the necessary information of analyzed works. During the query relaxation process, all analyzed works access the RDF dataset instances to evaluate relaxed queries and progressively obtain k-relevant answers. Dataset statistics are used to calculate the similarity of queries based on information content in [16, 20]. This technique allows to rank queries so that unrelevant queries are ignored. Thus, it allows to avoid evaluating an important number of relaxed queries.

Query shapes. Third column of Table 4, shows that most of the works are able to relax the three main query structures: star, chain, and composite queries. All works relax star queries. [12, 15] focus exclusively on subject star-shaped queries. [8] is limited to the relaxation of single triple patterns. But the extension of this work in [11] also proposes to relax regular path queries.

Dataset summaries are also used to limit the communication overhead in a federated environment in [20], where a federation of SPARQL endpoints participate in evaluating relaxed (federated) queries. In [12], the RDF dataset is used to build Bayesian networks which help in estimating the similarity measure of every relaxed query. All approaches of the first set of works, except for CADER, need the dataset ontology. CADER analyzes the original triple patterns of a user query to identify the maximal subsets of non failing triple patterns.

Concerning the second part of the contributions, based on the RDF dataset, [13] produces probabilistic language models. [15] uses the RDF dataset to map every pair values with distance functions. During the relaxation process, [17] access the RDF dataset to define partitions. As an option, [17] can use the dataset ontology. None of these works use dataset statistics.

Terms relaxed. The fourth column of Table 4 shows that subjects are relaxed by almost all analyzed works. [12, 15, 19] do not relax subjects. Predicates are relaxed by almost all works. All analyzed works relax objects of triple patterns. [15] relaxes only objects with its framework of mapping functions with similarity distances among object values. [18] does not relax queries ontologically, it only eliminates failing triple patterns.

5.2. *Query relaxation analysis*

Ontology-based query relaxation is made dynamically as an online process. Even if the relaxation possibilities can be huge, O-MBS and BR ([14, 16]) proposed optimized strategies to prune the query relaxation lattice. Furthermore, the ontology-based relaxation proposed in [19], shows that it is also possible to relax using SHACL constraints. [19] also demonstrates that weighting triple patterns in the query similarity calculation can lead to obtaining more efficiently the k-relevant answers at earlier levels of the 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 distance functions are costly. Distance 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 no impact on the computation of relaxed queries. Only dataset statistics need 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.

Ontology relaxation. Second column of Table 5 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 hierarchy. Only the contributions proposed in [8, 9, 11, 12] mention using typing relaxation using the range and the 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, [8, 9, 11] do not use query similarity. Most of the relaxation approaches use type and property relaxations except the contributions that focus on similarity of instances [13, 15] and the one that only consists of eliminating triple patterns [18].

Simple relaxation. Third column of Table 5 indicates that all the analyzed aproaches use simple relaxation except [19]. 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 a triple pattern with only variables.

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 [8, 9, 11]	✓	✓	✓	✓	✗	✗	✓	Relaxation distance
Ranking model based on Bayesian networks to efficiently relax star queries [12]	✓	✓	✓	✓	✗	✗	✗	Relaxation distance + similarity metric
BFSR [14]	✓	✓	✗	✓	✗	✗	✗	Relaxation distance + similarity metric
OBFSR [14]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric
BR [14]	✓	✓	✗	✓	✗	✓	✓	Relaxation distance + similarity metric
MBS [16]	✓	✓	✗	✓	✗	✓✗	✗	Relaxation distance + similarity metric
O-MBS [16]	✓	✓	✗	✓	✗	✓✗	✗	Relaxation distance + similarity metric
F-MBS [16]	✓	✓	✗	✓	✗	✓	✗	Relaxation distance + similarity metric
CADER [18]	✗	✗	✗	✓	✗	✓	✗	✗
Query relaxation for portable brick-based applications [19]	✓	✗	✗	✗	✗	✗	✗	Relaxation distance + similarity metric
FLiQue [20]	✓	✓	✗	✓	✗	✓✗	✗	Relaxation distance + similarity metric
Statistical language model relaxation [13]	✗	✗	✗	✓	✓	✗	✗	Statistical language-model based ranking
Relaxations of heterogenous resources descriptions [15]	✗	✗	✗	✓	✓	✗	✗	Normalized Euclidean distance/edit distances/concurrence algorithm
Partitioning-based relaxation [17]	✓	✓	✗	✓	✓	✗	✓	Relaxation distance and symbolic similarity

Table 5
Comparison of query relaxation approaches.

Relaxation based on similarity of instances. Fourth column of Table 5 higlights the second set of analyzed works that focus on the similarity of intances. Queries are relaxed based on the similarity between the dataset 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 5 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 for performance reasons. [14, 16] studied the failure causes of relaxed queries. MBS and O-MBS can discover some relaxed queries that do not give answers without evaluating 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 5 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 dataset 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 [8, 9, 11, 17] and the BR algorithm [14] make efforts to reduce redundancy. [8, 9, 11] avoid redundancy by considering the logical subsumption between queries. The lazy join algorithm proposed in [17] also results in calculating non-redundant answers by optimizing joins. The BR algorithm [14] 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, almost all analyzed works use techniques for query ranking. This is shown in the last column of Table 5. 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 3.2.3). [8, 9, 11] rank queries according to their relaxation distance. Similarity of queries based on information content is used by [12, 14, 16, 19, 20]. [19] extends the similarity calculation by proposing simple similarity measures for relaxations using SHACL constraints (for domain and range constraints) and when a predicate is relaxed into a property path (into a predicate+). Other works employ different ways of computing similarity, such as word embeddings methods [42], symbolic similarity [17], numeric distance functions [15], and JS-divergence between statistical models [13]. CADER [18] does not consider ranking queries. It just discovers the set of relaxed queries, without any order.

5.3. Experimental evaluations

SPARQL query relaxation has been used in solving real-world problems. Among the surveyed works, [15] addresses the need for approximate answers in a real scenario, specifically in police post-incident analysis. In this context, data is gathered and integrated for providing a more complete and insightful view of the incident or scenario under investigation. In addition, [19] seeks to enable interoperability and efficiency in smart buildings and building management systems. [19] allows applications to dynamically discover and interact with building resources, optimizing energy management, fault detection, occupant comfort, and scalability while reducing the need for hard-coded references to specific data sources. However, the remaining works do not focus on specific real-world scenarios.

Concerning the experimental evaluations reported in analyzed works, they vary in the used dataset size, the number of experimented queries, and the size of the queries (number of triple patterns in a BGP). Dataset used across analyzed works can be classified into three distinct dimensions, reflecting the diverse scales and issues of RDF data they encompass. The size of the dataset crossed with the potential relaxation allowed by the ontology poses important challenges of scalability. Exploring all the possibilities of relaxation of the query relaxation lattice is not possible but the limits of instances and ontology-hierarchies makes algorithms to reach an end point.

Small-scale datasets EADS, MONDIAL, and the Brick dataset contain a modest number of RDF triples, enabling controlled experiments and specific evaluations.

Medium-scale datasets A number of reviewed works chose to evaluate query relaxation on medium-scale datasets, including LUBM10, LibraryThing, and subsets of the IMDB dataset. These datasets strike a balance between complexity and manageability, presenting a realistic representation of query scenarios in various domains.

Large-scale datasets Our survey also covers works that tackle the challenges of large-scale RDF datasets. Datasets such as LUBM100, DBpedia, and LargeRDFBench contain extensive RDF triples, allowing for evaluating the scalability and efficiency of query relaxation techniques under substantial data volumes.

Table 6 shows the comparison of the relaxation approaches based on the experimental evaluations. [8, 9, 11] 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. The second column of Table 6 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 [36]. Besides using LUBM for ontology relaxation, [17] uses MONDIAL to study the performance of the proposed approach in presence of multi-valued properties. The brick-based relaxation [19] uses a different dataset concerning the brick-based building as it targets particular applications. 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 interesting query relaxation. At the time, LargeRDFBench was the only benchmark for federated queries. [13] tested their approaches on parts of the LibraryThing and IMDB datasets. These datasets were automatically parsed and converted into RDF triples in this work. [15] uses an existing a structured dataset describing car instances. The dataset was modelled and transformed into RDF triples by the authors using the Hepp's Vehicle Sales Ontology[27].

Number of queries and size of queries. The third and fourth columns of Table 6 show heterogeneous number of queries and number of triple 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 the experimental goals. CADER [13] 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. [10] experimented with five subject star-shaped queries having from three to four triple patterns. [14] and [16] used seven LUBM queries. [14] 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.

[16] 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. [19] was experimented over small and simple queries (up to 8 triple patterns). FLiQue [20] 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 numerous variables in both subjects and objects, with very few classes in objects. (they are all star and composite queries). Thus the relaxation opportunities are mostly concentrated in the properties.

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

[13, 15] 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). [13] defined 40 evaluation queries for each dataset. Top-5 relaxed queries were generated for each evaluation query. [15] defined three vague queries run against the vehicles dataset where five relaxations were also generated for each query.

The partitioning-based approach [17] 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 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 in the absence of ontology 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.

Comparison to other works. It is hard to position the efficiency of analyzed works due to the lack of availability of source codes and experimental prototypes. Indeed, analyzed works barely position experimentaly their contributions to state-of-the-art works. CADER was compared experimentally with algorithms proposed in [36] (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 [16], were compared as a whole with the BFSR algorithm [14]. Experiments revealed that BFSR executes more queries until finding k answers than the proposed algorithms in [16]. 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 [16][28],[19][29], [20][30], and [17][31].

5.4. Impact of SPARQL query relaxation over RDF reification

Several methods exist to define statement-level annotations. In this article we focus on standard reification, named graphs, n-ary relations, singleton properties, and RDF-star (cf. Section 3.3). In this section, we analyse the feasability of applying analyzed query relaxation approaches to RDF datasets using these reification methods under different perspectives.

Analyzing the syntax support, RDF-star, being a recent proposal, is not currently supported by existing relaxation proposals. Indeed, the nested capability of RDF-star and named graphs is not considered in query relaxation. The other reification methods can be syntactically supported but triple patterns querying metadata will be relaxed in the same manner as triple patterns querying data. However, relaxing classes and properties that specify the reification method, such as rdf:singletonPropertyOf, rdf:Statement, rdf:subject, rdf:object, rdf:predicate and the n-ary relation ex:Enroll_relation in Listing 1, is not meaningful. Moreover, the query shape of typical queries for standard reification are star-shaped queries, and for n-ary relations and singleton properties, they are composite queries with a subject star-shaped part. This enables the application of most relaxation methods to these reification models. Table 7 shows for each existing relaxation method and reification model, which combinations are technically applicable with minimal modifications.

Uncertain knowledge graphs have been addressed in the work [43]. They proposed algorithms in the context of empty answers. Their approach explains the reasons for the failure. There, the only metadata provided is a trust score related to each triple. They applied their algorithms over three reification models: standard reification, named graphs, and N-Quads. The experiments showed that query execution times are significantly longer on the named graph and reification implementations in comparison with the quad filter implementation

The following paragraphs analyze these and other comparative aspects in more details.

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

[29] https://github.com/anandkp92/relaxed-brick-queries/

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

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

Contributions	**Used dataset**	**Number of queries**	**Size of queries**	**Comparison with other works**	**Available source code**
Extension of the OPTIONAL clause with a RELAX clause [8, 9, 11]	-	-	-	✗	✗
Ranking model based on Bayesian networks [12]	LUBM (600K triples)	5 queries	3 to 4 triple patterns	✗	✗
BFSR, OBFSR, BR [14]	LUBM (10M triples)	7 queries	2 to 5 triple patterns	✗	✗
MBS, O-MBS, F-MBS [16]	LUBM (17M to 167M triples)	7 queries	1 to 15 triple patterns	Compared with BFSR [14]	✓
CADER [18]	LUBM and DBpedia (65M to 2B triples)	21 queries for DBpedia, 197 queries for LUBM	2 to 15 triple patterns	Compared with LBA and MBA [36]	✗
Query relaxation for portable brick-based applications [19]	Brick dataset (5 to 8K triples)	8 queries	2 to 8 triple patterns	✗	✓
FLiQue [20]	LargeRDFBench (> 1B triples)	32 federated queries	2 to 12 triple patterns	✗	✓
Statistical language model relaxation [13]	2 datasets generated from LibraryThing community and IMDB (700K triples)	40 queries for each dataset	1 to 4 triple patterns	✗	✗
Relaxations of heterogenous resources descriptions [15]	Vehicles dataset of the EADS project (50K triples)	3 queries	2 to 3 triple patterns	✗	✗
Partitioning-based relaxation [17]	MONDIAL (12K triples) and LUBM (1.3M to 13M triples)	7 queries for each dataset	Small queries (up to 21 triple patterns), large queries (up to 1505 triple patterns)	✗	✓

Table 6
Comparison of experimental evaluations.

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 a triple pattern or a graph.

Impact of query shape. All relaxation approaches can relax star-shaped queries. Thus, all of them can deal with standard reification where queries are typical star-shaped ones with the subject variable searching for instances that are of type Statement. The n-ary relations and singleton properties induce composite queries. Except for [12] and [15], all relaxation works are able to relax composite querie where objects and predicates can be relaxed.

Relaxation over triple patterns querying metadata. Relaxation based on ontologies over queries including metadata constraints will apply relaxation over the terms related to the reification model or the metadata value itself. The ontology hierarchy of classes and properties used in annotations is not rich and relaxing them will not have any positive effect in the relaxation process. It has no sense in logically relaxing terms like rdf:singletonPropertyOf, rdf:type, rdf:Statement, rdf:subject, rdf:object, rdf:predicate and the n-ary relation ex:Enroll_relation and the property ex:enroll_value in Listing 1. Simple relaxation, where a variable would replace terms (instance or the constant) in the subject or the object of a metadata triple pattern would have a better effect. Using relaxation approaches based on similarity of instances will be more efficient over the annotation values. These approaches will relax values giving empty answers with similar or closer values. We notice, however, that none of the works use range of values. Using range of numeric values would be an interesting relaxation for annotation values.

Impact of query size. Querying data and its metadata produces, in general, queries with numerous triple patterns and joins. As the number of triple patterns increases, the complexity of the computation of relaxed queries increases. Some works can deal with BGPs having until 15 triple patterns. [17] experimentd with a very large number of triple patterns but only with the similarity of instances approach, not the ontology-based relaxation. Ontology-based relaxation poses serious challenges when the ontology hierarchy is rich. Moreover, 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 because they do not add terms and triple patterns exclusively for the reification syntax. Unfortunately, there is no relaxing approach designed either for SPARQL-star or named graph queries.

Impact of the metadata type. The 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.

Impact of the 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 dataset. In general, adding statement-based annotations to RDF graphs increases the dataset volume. Except for named graphs and RDF-star, other reificaton models need several triples to include a statement-based annotation. This volume will impact the first phase of relaxation approaches based on similarity of instances. Besides, the shape of the RDF graph depends on the reification model (see Figure 5). Thus, approaches like the one based on statistical language model relaxation [13] or the partitioning-based relaxation [17] would be seriously impacted.

On the other side, ontology-based relaxation approaches are independent of the size of the dataset during relaxation process, 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. At the evaluation level, the relaxation works [16, 20] that use dataset statistics will also not be affected at this level.

Regarding the evaluation of relaxed queries, depending on the query engine and reification model, the size of the dataset can cause performance issues (see Section 3.3.6).

After surveying existing SPARQL query relaxation strategies, we cannot assert that a relaxation approach can be directly applied to RDF reification. Several adaptations should be done. Each analyzed relaxation approach needs to be modified to understand the semantics of reification. The relaxation approaches could be enhanced to distinguish data and metadata in a query and identify reification patterns. This should be accompanied with understanding the semantics of reification patterns and treat it differently from regular triples. Particular attention should be paid to the chain part of composite queries because the relation of the data and the metadata parts of the query should not be broken. This could involve developing specialized similarity measures or scoring mechanisms for reification triples. This would ensure that the reification semantics are better retained during relaxation.

6. 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. Here is a summary of our observations. (1) Logical relaxation over the terms related to the reification model or the metadata value itself has in general no sense. That is, logically relaxing terms like rdf:singletonPropertyOf, rdf:type, rdf:Statement,

Contributions	Standard reification	Named graphs	N-ary relations	Singleton properties	RDF-star
Extension of the OPTIONAL clause with a RELAX clause [8, 9, 11]	✓	✗	✓	✓	✗
Ranking model based on Bayesian networks [12]	✓	✗	✗	✗	✗
BFSR, OBFSR, BR [14]	✓	✗	✓	✓	✗
MBS, O-MBS, F-MBS [16]	✓	✗	✓	✓	✗
CADER [18]	✓	✗	✓	✓	✗
Query relaxation for portable brick-based applications [19]	✓	✗	✓	✓	✗
FLiQue [20]	✓	✗	✓	✓	✗
Statistical language model relaxation [13]	✓	✗	✓	✓	✗
Relaxations of heterogenous resources descriptions [15]	✓	✗	✗	✗	✗
Partitioning-based relaxation [17]	✓	✗	✓	✓	✗

Table 7

Applicability of existing relaxation works with respect to reification models.

rdf:subject, rdf:object, rdf:predicate does not generate relevant answers. (2) Metadata values are not taken into account in the query ranking function that allows gathering the k-relevant answers closest to the original query. In ontology-based approaches, object-values are replaced by variables. (3) Current relaxation techniques are defined for equality of values but metadata constraints can include intervals of values in a range. Relaxation approaches based on similarity of instances can be efficient over annotation values but in general they do not consider ranges of values. (4) The representation of different RDF reification models leads to different forms of the RDF graph. Graphs representing instances with reification vary and will be consequential to approaches based on the similarity of instances that use graph-based approaches to calculate instances' similarity. (5) Syntactically, RDF-star and named graphs are not supported by current query relaxation approaches. Indeed, the nesting aspect of these reification models is not considered.

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 does not have the same goal as relaxing other triple patterns. However, the joins between the data and metadata part should be preserved (not broken down).

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. If as we suggest, the terms related to the reification models are not logically relaxed, the query relaxation lattice will decrease. New relaxation approaches should take care about not breaking the relation of the metadata part of the query from the data part. 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. Experimental evaluations to compare existing works will be done in the future works. Among the instance-based works, particular interest lies in evaluating the partitioning-based approach, especially considering its non-specific targeting of particular application scenarios or query shapes. While for the ontology-based works, [14, 16] stand out as particularly interesting candidates for comparison. Their focus on composite queries and effective optimizations makes them compelling choices for evaluation. Regarding the datasets, it would be motivating to compare these works using Wikidata since it is a diverse and extensive knowledge base, covering a wide range of domains. Both conjuctive and non conjunctive queries could be the target of these experimental evaluations.

Finally, current works should also be extended to cover the syntax and the nesting aspect of named graphs and RDF-star. Besides, existing query engines can deal efficiently with named graphs and extensions taking into account this representation can be effective, but that is not the case for RDF-star. SPARQL query engines should be improved to make possible query relaxation over RDF-star datasets.

7. 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.

8. 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] P. Godfrey, Minimization in cooperative response to failing database queries, *International Journal of Cooperative Information Systems (IJICIS)* **6**(02) (1997), 95–149.

[3] P. Resnik, Using information content to evaluate semantic similarity in a taxonomy, in: *International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 1, 1995, pp. 448––453.

[4] F. Manola, E. Miller, B. McBride et al., RDF primer, *W3C recommendation* **10**(1–107) (2004), 6.

[5] J.J. Carroll, C. Bizer, P. Hayes and P. Stickler, Named graphs, *Journal of Web Semantics (JWS)* **3**(4) (2005), 247–267.

[6] V. Nguyen, O. Bodenreider and A. Sheth, Don't like RDF reification? Making statements about statements using singleton property, in: *International Conference on World Wide Web (WWW)*, 2014, pp. 759–770.

[7] O. Hartig, Foundations of RDF* and SPARQL*:(An alternative approach to statement-level metadata in RDF), in: *Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW)*, Vol. 1912, CEUR Workshop Proceedings, 2017.

[8] 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.

[9] C.A. Hurtado, A. Poulovassilis and P.T. Wood, Query relaxation in RDF, *Journal on Data Semantics X* **4900** (2008), 31–61.

[10] 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.

[11] 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.

[12] H. Huang and C. Liu, Query relaxation for star queries on RDF, in: *International Conference on Web Information Systems Engineering (WISE)*, Vol. 6488, Springer, 2010, pp. 376––389.

[13] 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.

[14] H. Huang, C. Liu and X. Zhou, Approximating query answering on RDF databases, *World Wide Web (WWW)* **15** (2012), 89–114.

[15] 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.

[16] 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.

[17] 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.

[18] W. Mebrek, B. Raddaoui and M. Albilani, On relaxing failing queries over RDF databases, in: *International Conference on Big Data*, IEEE, 2019, pp. 115–124.

[19] I.L. Bennani, A.K. Prakash, M. Zafiris, L. Paul, C.D. Roa, P. Raftery, M. Pritoni and G. Fierro, Query relaxation for portable brick-based applications, in: *International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys)*, 2021, pp. 150–159.

[20] 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.

[21] S. Ross, *A first course in probability*, Macmillan, 1976.

[22] F. Zhang, K. Wang, Z. Li and J. Cheng, Temporal data representation and querying based on RDF, *IEEE access* **7** (2019), 85000–85023.

[23] A. Pugliese, O. Udrea and V. Subrahmanian, Scaling RDF with time, in: *International Conference on World Wide Web (WWW)*, 2008, pp. 605–614.

[24] P. Bosc, O. Pivert and H. Prade, An uncertain database model and a query algebra based on possibilistic certainty, in: *International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, IEEE, 2010, pp. 63–68.

[25] D. Hernández, A. Hogan and M. Krötzsch, Reifying RDF: What works well with wikidata?, *International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)* **1457** (2015), 32–47.

[26] S. Sen, D. Katoriya, A. Dutta and B. Dutta, RDFM: An alternative approach for representing, storing, and maintaining meta-knowledge in web of data, *Journal of Expert Systems with Applications* **179** (2021), 115043.

[27] O. Udrea, D.R. Recupero and V. Subrahmanian, Annotated rdf, *ACM Transactions on Computational Logic (TOCL)* **11**(2) (2010), 1–41.

[28] N. Noy, A. Rector, P. Hayes and C. Welty, Defining n-ary relations on the semantic web, *W3C working group note* (2006).

[29] 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.

[30] M. Kieffer, G. Fakih and P. Serrano-Alvarado, Evaluating Reification with Multi-valued Properties in a Knowledge Graph of Licensed Educational Resources, in: *International Conference on Semantic Systems (SEMANTiCS)*, IOS Press, 2023, pp. 94–109.

[31] A. Iglesias-Molina, K. Ahrabian, F. Ilievski, J. Pujara and Ó. Corcho, Comparison of Knowledge Graph Representations for Consumer Scenarios, in: *International Semantic Web Conference (ISWC)*, Springer, 2023, pp. 271–289.

[32] 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.

[33] G. Abuoda, C. Aebeloe, D. Dell Aglio, A. Keen and K. Hose, StarBench: Benchmarking RDF-star Triplestores, in: *Workshop on Storing, Querying and Benchmarking Knowledge Graphs (QuWeDa)*, 2023.

[34] G.F. Cooper and E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, *Machine learning* **9** (1992), 309–347.

[35] Y. Guo, Z. Pan and J. Heflin, LUBM: A benchmark for OWL knowledge base systems, *Journal of Web Semantics (JWS)* **3** (2005), 158–182.

[36] 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.

[37] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL). W3C, 2017. https://www.w3.org/TR/shacl/.

[38] B. Moreau, P. Serrano-Alvarado, M. Perrin and E. Desmontils, Modelling the compatibility of licenses, in: *Extended Semantic Web Conference (ESWC)*, Vol. 11503, Springer, 2019, pp. 255–269.

[39] M. Saleem, A. Potocki, T. Soru, O. Hartig and A.-C.N. Ngomo, CostFed: Cost-based query optimization for SPARQL endpoint federation, in: *International Conference on Semantic Systems (SEMANTICS)*, Vol. 137, Elsevier, 2018, pp. 163–174.

[40] 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.

[41] B. Ganter and R. Wille, *Formal concept analysis: mathematical foundations*, Springer Science & Business Media, 2012.

[42] 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.

[43] I. Dellal, S. Jean, A. Hadjali, B. Chardin and M. Baron, Query answering over uncertain RDF knowledge bases: explain and obviate unsuccessful query results, *Knowledge and Information Systems* **61**(3) (2019), 1633–1665.

Appendix A. Supplemental material

A.1. Example dataset

Listing 3: Dataset used as running example.

```
1  @prefix ex:   <http://example.org/> .
2  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5
6
7  # Triples about Data
8  ex:Databases     rdf:type         ex:Course ;
9                   ex:difficulty    "hard".
10 ex:Semantic_Web  rdf:type         ex:Course ;
11                  ex:difficulty    "easy".
12 ex:John          rdf:type         ex:Student ;
13                  ex:enrolledIn    ex:Semantic_Web ;
14                  ex:friendOf      ex:Sam ;
15                  foaf:knows       ex:Patricia .
16 ex:Maya          rdf:type         ex:Student ;
17                  ex:enrolledIn    ex:Semantic_Web ;
18                  foaf:knows       ex:Peter ;
19                  foaf:knows       ex:Patricia.
20 ex:Maria         rdf:type         ex:Student ;
21                  ex:enrolledIn    ex:Databases ;
22                  ex:friendOf      ex:Peter;
23                  foaf:knows       ex:Patricia.
24 ex:Patricia      rdf:type         ex:Teacher ;
25                  ex:worksIn       ex:Nantes_Universite ;
26                  ex:teaches       ex:Semantic_Web , ex:Databases .
27 ex:Peter         rdf:type         ex:Person;
28                  foaf:knows       ex:Alice.
29 ex:Sam           rdf:type         ex:Person;
30                  foaf:knows       ex:Alice.
31 ex:Alice         rdf:type         ex:Person.
32
33 # Triples about Metadata
34
35 # Metadata about the triple in line 12
36 ex:S100   rdf:type          rdf:Statement ;
37           rdf:subject       ex:John ;
38           rdf:predicate     ex:enrolledIn ;
39           rdf:object        ex:Semantic_Web ;
40           ex:enrolldate     "2022-09-12"^^xsd:date ;
41           ex:enrollpayment  "Cash"^^xsd:string.
42
43 # Metadata about the triple in line 16
44 ex:S300   rdf:type          rdf:Statement ;
45           rdf:subject       ex:Maya ;
46           rdf:predicate     ex:enrolledIn ;
47           rdf:object        ex:Semantic_Web;
48           ex:enrolldate     "2023-09-13"^^xsd:date ;
49           ex:enrollpayment  "Credit␣card"^^xsd:string .
50
51 # Metadata about the triple in line 20
52 ex:S200   rdf:type          rdf:Statement ;
53           rdf:subject       ex:Maria ;
54           rdf:predicate     ex:enrolledIn ;
55           rdf:object        ex:Databases ;
56           ex:enrolldate     "2023-09-13"^^xsd:date ;
57           ex:enrollpayment  "Cash"^^xsd:string.
```

A.2. Example ontology

Listing 4: Ontology used for running example.

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .


# Define Classes
ex:Student rdf:type rdfs:Class .
ex:Person  rdf:type rdfs:Class .
ex:Teacher rdf:type rdfs:Class .
ex:Course  rdf:type rdfs:Class .
rdf:Statement rdf:type rdfs:Class.

# Define Object Properties
ex:attends      rdf:type owl:ObjectProperty ;
        rdfs:domain ex:Person;
        rdfs:range ex:Course.
ex:enrolledIn rdf:type owl:ObjectProperty ;
        rdfs:domain ex:Student.
ex:friendOf rdf:type owl:ObjectProperty .
ex:knows  rdf:type owl:ObjectProperty ;
  rdfs:domain ex:Person ;
      rdfs:range ex:Person .
ex:classmates     rdf:type owl:ObjectProperty ;
     rdfs:domain ex:Person ;
     rdfs:range ex:Person .
ex:teaches rdf:type owl:ObjectProperty ;
     rdfs:domain ex:Teacher;
     rdfs:range  ex:Course.


# Define Data Property
ex:enrolldate rdf:type owl:DatatypeProperty ;
        rdfs:domain rdf:Statement;
        rdfs:range xsd:date.
ex:difficulty rdfs:domain ex:Course;
              rdfs:range xsd:string.
ex:enrollpayment rdf:type owl:DatatypeProperty ;
        rdfs:domain rdf:Statement;
        rdfs:range xsd:string.


# Define classes hierarchy
ex:Student     rdfs:subClassOf ex:Person.
ex:Teacher     rdfs:subClassOf ex:Person.
rdf:Statement rdfs:subClassOf rdfs:Resource.

# Define properties hierarchy
ex:friendOf rdfs:subPropertyOf  foaf:knows.
ex:teaches rdfs:subPropertyOf  ex:attends.
ex:enrolledIn rdfs:subPropertyOf  ex:attends.


# RDFS ontology
rdfs:Resource rdf:type rdfs:Class .

rdfs:Class rdf:type rdfs:Class ;
  rdfs:subClassOf rdfs:Resource .
```

A.3. Closure of the example dataset

Listing 5: Closure computed for the running example dataset.

```

@prefix ex:   <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

ex:Student  rdf:type      rdfs:Resource , rdfs:Class ;
        rdfs:subClassOf   ex:Person .

ex:Peter  rdf:type  rdfs:Class , rdfs:Resource , ex:Person ;
        foaf:knows  ex:Alice .

ex:Semantic_Web  rdf:type  rdfs:Resource , rdfs:Class , ex:Course ;
        ex:difficulty  "easy" .

ex:S300  rdf:type           rdfs:Class , rdfs:Resource , rdf:Statement ;
        rdf:object          ex:Semantic_Web ;
        rdf:predicate       ex:enrolledIn ;
        rdf:subject         ex:Maya ;
        ex:enrolldate       "2023-09-13"^^xsd:date ;
        ex:enrollpayment    "Credit_card" .

rdfs:Class  rdf:type      rdfs:Resource , rdfs:Class ;
        rdfs:subClassOf   rdfs:Resource .

rdfs:Resource  rdf:type   rdfs:Resource , rdfs:Class .

ex:Maya  rdf:type       rdfs:Class , rdfs:Resource , ex:Person , ex:Student ;
        ex:attends      ex:Semantic_Web ;
        ex:enrolledIn   ex:Semantic_Web ;
        foaf:knows      ex:Patricia , ex:Peter .

rdf:Statement  rdf:type   rdfs:Resource , rdfs:Class ;
        rdfs:subClassOf   rdfs:Resource .

ex:Person  rdf:type  rdfs:Resource , rdfs:Class .

ex:Maria  rdf:type      rdfs:Class , rdfs:Resource , ex:Person , ex:Student ;
        ex:attends      ex:Databases ;
        ex:enrolledIn   ex:Databases ;
        ex:friendOf     ex:Peter ;
        foaf:knows      ex:Peter , ex:Patricia .

ex:S200  rdf:type           rdfs:Resource , rdfs:Class , rdf:Statement ;
        rdf:object          ex:Databases ;
        rdf:predicate       ex:enrolledIn ;
        rdf:subject         ex:Maria ;
        ex:enrolldate       "2023-09-13"^^xsd:date ;
        ex:enrollpayment    "Cash" .

ex:Course  rdf:type  rdfs:Resource , rdfs:Class .

ex:Teacher  rdf:type      rdfs:Resource , rdfs:Class ;
        rdfs:subClassOf   ex:Person .

ex:Patricia  rdf:type  rdfs:Resource , ex:Person , rdfs:Class , ex:Teacher ;
        ex:attends  ex:Databases , ex:Semantic_Web ;
        ex:teaches  ex:Databases , ex:Semantic_Web ;
        ex:worksIn  ex:Nantes_Universite .

ex:Sam  rdf:type     rdfs:Resource , rdfs:Class , ex:Person ;
        foaf:knows  ex:Alice .

ex:S100  rdf:type           rdfs:Resource , rdfs:Class , rdf:Statement ;
        rdf:object          ex:Semantic_Web ;
        rdf:predicate       ex:enrolledIn ;
        rdf:subject         ex:John ;
```

```
            ex:enrolldate       "2022-09-12"^^xsd:date ;
            ex:enrollpayment   "Cash" .

    ex:Databases   rdf:type   rdfs:Resource , rdfs:Class , ex:Course ;
            ex:difficulty   "hard" .

    ex:Alice   rdf:type   rdfs:Resource , rdfs:Class , ex:Person .

    ex:John   rdf:type        rdfs:Class , rdfs:Resource , ex:Person , ex:Student ;
            ex:attends      ex:Semantic_Web ;
            ex:enrolledIn   ex:Semantic_Web ;
            ex:friendOf     ex:Sam ;
            foaf:knows      ex:Sam , ex:Patricia .

    ex:classmates   rdf:type   owl:ObjectProperty ;
            rdfs:domain   ex:Person ;
            rdfs:range    ex:Person .

    ex:attends   rdf:type   owl:ObjectProperty ;
            rdfs:domain   ex:Person ;
            rdfs:range    ex:Course .

    ex:difficulty   rdfs:domain   ex:Course ;
            rdfs:range    xsd:string .

    ex:enrollpayment   rdf:type   owl:DatatypeProperty ;
            rdfs:domain   rdf:Statement ;
            rdfs:range    xsd:string .

    ex:knows   rdf:type    owl:ObjectProperty ;
            rdfs:domain   ex:Person ;
            rdfs:range    ex:Person .

    ex:enrolldate   rdf:type   owl:DatatypeProperty ;
            rdfs:domain   rdf:Statement ;
            rdfs:range    xsd:date .

    ex:teaches   rdf:type          owl:ObjectProperty ;
            rdfs:domain           ex:Teacher ;
            rdfs:range            ex:Course ;
            rdfs:subPropertyOf    ex:attends .

    ex:enrolledIn   rdf:type      owl:ObjectProperty ;
            rdfs:domain           ex:Student ;
            rdfs:subPropertyOf    ex:attends .

    ex:friendOf   rdf:type        owl:ObjectProperty ;
            rdfs:subPropertyOf    foaf:knows .
```

A.4. Dataset statistics

Class	Number of entities
ex:Student	3
ex:Teacher	1
ex:Course	2
ex:Person	7
rdf:Statement	3
rdfs:Resource	19
Total	54

Table 8

Statistics of classes.

Property	Number of triples
ex:enrolledIn	3
ex:teaches	2
ex:attends	5
ex:friendOf	2
foaf:knows	8
rdf:type	61
rdf:subject	3
rdf:object	3
rdf:predicate	3
ex:enrolldate	3
ex:difficulty	2
Total	122

Table 9

Statistics of properties.

Relaxed queries	Similarity measure
Q_{14}	0.87
Q_5	0.81
Q_3	0.8
Q_9	0.72
Q_1	0.71
Q_2, Q_4, Q_6, Q_7, Q_8, Q_{10} , Q_{11}, Q_{12}, Q_{13}, Q_{15}, Q_{16}, Q_{17}	0.67

Table 10

Similarity of relaxed queries.

A.5. Relaxed queries of the first level of the relaxation lattice.

Listing 6: Relaxed queries (first level of the relaxation lattice) and their similarity with query Q.

```
  # Query Q:

  PREFIX  ex:    <http://example.org/>
  PREFIX  rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  PREFIX  owl:   <http://www.w3.org/2002/07/owl#>
  PREFIX  xsd:   <http://www.w3.org/2001/XMLSchema#>
  PREFIX  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
  PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>

  SELECT ?student
  WHERE
  {?teacher    rdf:type              ex:Teacher;                  #tp1     Querying data
               ex:teaches            ex:Semantic_Web.             #tp2
   ?student    rdf:type              ex:Student;                  #tp3
               ex:friendOf/foaf:knows  ex:Alice;                  #tp4
               foaf:knows              ?teacher.                  #tp5
   ?statement  rdf:type              rdf:Statement;               #tp6     Querying metadata
               rdf:subject           ?student;                    #tp7
               rdf:object            ex:Semantic_Web;             #tp8
               rdf:predicate         ex:enrolledIn;               #tp9
               ex:enrolldate         "2023-09-13"^^xsd:date.      #tp10
  }

# No result

# ------------------------------
# Query Q1:

SELECT  ?student
WHERE
  { ?teacher   rdf:type    ex:Person ;                  #relaxed (type relaxation)
               ex:teaches  ex:Semantic_Web .
    ?student   rdf:type    ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student   foaf:knows      ?teacher .
    ?statement   rdf:type      rdf:Statement ;
               rdf:subject     ?student ;
```

```
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.7142857142857142
# No result


# ------------------------------
# Query Q2:

SELECT ?student
WHERE
  { ?teacher   ?p1          ex:Teacher ;                 #relaxed (simple relaxation)
              ex:teaches    ex:Semantic_Web .
    ?student   rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student   foaf:knows      ?teacher .
    ?statement   rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result


# ------------------------------
# Query Q3:

SELECT ?student
WHERE
  { ?teacher   rdf:type     ex:Teacher ;
              ex:attends    ex:Semantic_Web .          #relaxed (property relaxation)
    ?student   rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student   foaf:knows      ?teacher .
    ?statement   rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.7999999999999999
# No result


# ------------------------------
# Query Q4:

SELECT ?student
WHERE
  { ?teacher   rdf:type     ex:Teacher ;
              ex:teaches    ?o1 .                      #relaxed (simple relaxation)
    ?student   rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student   foaf:knows      ?teacher .
    ?statement   rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q5:

SELECT ?student
WHERE
```

```
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Person .                #relaxed (type relaxation)
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows       ?teacher .
    ?statement  rdf:type       rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.8095238095238095
# No result

# ------------------------------
# Query Q6:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  ?p2          ex:Student .                #relaxed (simple relaxation)
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows       ?teacher .
    ?statement  rdf:type       rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q7:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ?o2 .              #relaxed (simple relaxation)
    ?student  foaf:knows       ?teacher .
    ?statement  rdf:type       rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q8:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  ?p3              ?teacher .              #relaxed (simple relaxation)
    ?statement  rdf:type       rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q9:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdfs:Resource ;            #relaxed (type relaxation)
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.7192982456140351
# No result

# ------------------------------
# Query Q10:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  ?p4           rdf:Statement ;            #relaxed (simple relaxation)
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q11:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdf:Statement ;
              ?p5             ?student ;                 #relaxed (simple relaxation)
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q12:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ?o3 ;                     #relaxed (simple relaxation)
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }
```

```
# Similarity: 0.6666666666666666
# One result (ex:Maria)

# ------------------------------
# Query Q13:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              ?p6             ex:Semantic_Web ;           #relaxed (simple relaxation)
              rdf:predicate   ex:enrolledIn ;
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q14:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              rdf:predicate   ex:attends ;               #relaxed (property relaxation)
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.8666666666666666
# No result

# ------------------------------
# Query Q15:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
              ?p7             ex:enrolledIn ;            #relaxed (simple relaxation)
              ex:enrolldate   "2023-09-13"^^xsd:date
  }

# Similarity: 0.6666666666666666
# No result

# ------------------------------
# Query Q16:

SELECT ?student
WHERE
  { ?teacher  rdf:type     ex:Teacher ;
              ex:teaches   ex:Semantic_Web .
    ?student  rdf:type     ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows      ?teacher .
    ?statement  rdf:type      rdf:Statement ;
              rdf:subject     ?student ;
              rdf:object      ex:Semantic_Web ;
```

```
                rdf:predicate  ex:enrolledIn ;
                ?p8            "2023-09-13"^^xsd:date   #relaxed (simple relaxation)
  }

# Similarity: 0.6666666666666666
# No result

# -----------------------------
# Query Q17:

SELECT ?student
WHERE
  { ?teacher  rdf:type    ex:Teacher ;
              ex:teaches  ex:Semantic_Web .
    ?student  rdf:type    ex:Student .
    ?student ex:friendOf/foaf:knows ex:Alice .
    ?student  foaf:knows    ?teacher .
    ?statement  rdf:type    rdf:Statement ;
                rdf:subject   ?student ;
                rdf:object    ex:Semantic_Web ;
                rdf:predicate  ex:enrolledIn ;
                ex:enrolldate  ?o4                #relaxed (simple relaxation)
  }

# Similarity: 0.6666666666666666
# One result   (ex:John)
```