Link Traversal Querying for a Diverse Web of Data

Jürgen Umbrich [a,*], Aidan Hogan [b], Axel Polleres [a] and Stefan Decker [c]
[a] *Vienna University of Economics and Business; Welthandelsplatz 1; 1020 Vienna; Austria*
E-mail: jueumb@gmail.com, axel.polleres@wu.ac.at
[b] *Dept. of Computer Science; Universidad de Chile; Blanco Encalada 2120; Santiago; Chile*
E-mail: ahogan@dcc.uchile.com
[c] *INSIGHT @ NUI Galway; National University of Ireland, Galway; Ireland*
E-mail: stefan.decker@deri.org

Abstract. Traditional approaches for querying the Web of Data often involve centralised warehouses that replicate remote data. Conversely, Linked Data principles allow for answering queries live over the Web by dereferencing URIs to traverse remote data sources at runtime. A number of authors have looked at answering SPARQL queries in such a manner; these *link-traversal based query execution* (LTBQE) approaches for Linked Data offer up-to-date results and decentralised (*i.e.*, client-side) execution, but must operate over incomplete dereferenceable knowledge available in remote documents, thus affecting response times and "recall" for query answers. In this paper, we study the recall and effectiveness of LTBQE, in practice, for the Web of Data. Furthermore, to integrate data from diverse sources, we propose lightweight reasoning extensions to help find additional answers. From the state-of-the-art which (1) considers only dereferenceable information and (2) follows `rdfs:seeAlso` links, we propose extensions to consider (3) `owl:sameAs` links and reasoning, and (4) lightweight RDFS reasoning. We then estimate the recall of link-traversal query techniques in practice: we analyse a large crawl of the Web of Data (the BTC'11 dataset), looking at the ratio of raw data contained in dereferenceable documents vs. the corpus as a whole and determining how much more raw data our extensions make available for query answering. We then stress-test LTBQE (and our extensions) in real-world settings using the FedBench and DBpedia SPARQL Benchmark frameworks, and propose a novel benchmark called *QWalk* based on random walks through diverse data. We show that link-traversal query approaches often work well in uncontrolled environments for simple queries, but need to retrieve an unfeasible number of sources for more complex queries. We also show that our reasoning extensions increase recall at the cost of slower execution, often increasing the rate at which results return; conversely, we show that reasoning aggravates performance issues for complex queries.

Keywords: Linked Data, SPARQL, RDFS, OWL, Semantic Web, RDF, Web of Data, Live Querying, Reasoning

1. Introduction

A rich collection of RDF data has been published on the Web as *Linked Data*, by governments, academia, industry, communities and individuals alike [34]. The resulting collective of interlinked contributions from a wide variety of Linked Data publishers has been dubbed the "*Web of (Linked) Data*": a novel corpus of structured data distributed across the entire Web, described using the Semantic Web standards and made available under the Linked Data principles.

An open challenge is how to query this novel Web of Data in an effective manner [5]. SPARQL [48]—the W3C standardised RDF query language—provides a powerful declarative means to formulate structured queries over RDF data. However, processing SPARQL queries over the Web of Data broaches many technical

*Corresponding author. E-mail: jueumb@gmail.com challenges. Traditional centralised approaches cache information from the Web of Data in local optimised indexes, executing queries over the replicated content. However, maintaining up-to-date coverage of a broad selection of the Web of Data is exceptionally costly; centralised caches of the Web of Data must settle for either limited coverage and/or for indexing some out-of-date information [38, 57, 60]. Users of centralised query engines will thus often encounter stale or missing results with respect to the current Web of Data [60].

Conversely, per Linked Data principles, the URIs that name resources (often) map through HTTP to the physical location of structured data about them (called *dereferencing*). The Web of Data itself can thus be viewed as a scale-free, decentralised database, consisting of millions of Web documents indexing structured data [32, 34]. Furthermore, agents can traverse and navigate the Web of Data through typed "RDF links" to discover related information [34, § 4.5].

This perspective suggests that queries can be answered directly over the Web of Data using HTTP, where a number of *live querying* approaches have recently been proposed that take a (SPARQL) query, find *query-relevant* sources on the Web of Data, and dynamically retrieve data from these sources for runtime query processing. In one such approach, Hartig *et al.* [29] followed the view of the Web of Data as a global database and proposed to rely on dereferenceable URIs and RDF links for discovering sources relevant to answer a SPARQL query. Later work by Hartig [27] calls this idea "*Link Traversal Based Query Execution*" (which we abbreviate to *LTBQE*).

However, in the LTBQE approach, each HTTP lookup can take seconds to yield content, many such lookups may be required, and subsequent lookups to the same remote server may need to be artificially delayed to ensure "polite" access (*i.e.*, to avoid inadvertent denial-of-service attacks). In the live querying scenario, remote sources are accessed while a user is waiting for answers to their queries: thus response times are often (necessarily) much slower when compared with centralised query engines operating over locally replicated content. Thus, a core challenge for LTBQE is to retrieve a minimal number of relevant sources while maximising the number of answers returned. Relatedly, the success of LTBQE is premised on the assumption that query relevant data about a resource can be found in its dereferenced document; as we show later, this assumption only partially holds.

Herein, we look at the performance and recall of LTBQE in practice. We also propose that lightweight reasoning extensions—specifically relating to `owl:sameAs` and RDFS semantics—can help to "'squeeze" additional answers from sources and to find additional query-relevant sources on the diverse Web of Data. We apply analysis over a large sample of the Web of Data to get insights into what ratio of raw data is available to LTBQE through dereferenceable principles vs. raw data in the entire corpus; we also analyse to what extent our proposed extensions make additional query-relevant data available. We then apply LTBQE and our extensions to three different SPARQL query benchmarks to stress-test how the approaches work in real-world, uncontrolled settings. In general, we find that LTBQE works best for simple queries that require traversing a handful ($\sim\leq 100$) sources. We also show that our reasoning extensions often help to find additional answers at the cost of increased query time, but can run into trouble when accessing data from domains such as DBpedia (which has a high fan-out of `owl:sameAs` and schema level links) and can exacerbate performance issues with complex queries.

Paper contributions and structure. This paper extends previous work [58] where we first introduced our reasoning extensions. Herein, we propose new mechanisms for collecting schema data, test additional benchmarks, and greatly extend discussion throughout. This paper is then structured as follows:

§ 2 We first present some background work in the area of Linked Data querying and reasoning.

§ 3 We present some formal preliminaries for RDF, Linked Data, SPARQL, RDFS and OWL.

§ 4 We reintroduce the LTBQE approach using concrete HTTP-level methods.

§ 5 We introduce *LiDaQ* (*Li*nked *Da*ta *Q*uery engine): our implementation of LTBQE with novel reasoning extensions and optimisations.

§ 6 We analyse a crawl of $\sim$7.4 m RDF/XML documents from the Web of Data (BTC'11), looking at issues of dereferenceability.

§ 7 We survey SPARQL benchmarks and how live-querying papers have evaluated their works. We then propose a new benchmark called QWalk.

§ 8 We test LTBQE and our reasoning extensions for three query benchmarks in native Web settings.

§ 9 We conclude with a summary of contributions and remarks on future directions.

2. Background and Related Work

Our work relates to research on querying over RDF data; more precisely, we focus on executing SPARQL queries over the Web of Data in a manner adhering to the Linked Data principles. A comprehensive and recent overview of existing approaches to query Linked Data was published by Hose *et al.* [37]. We similarly classify relevant query approaches into three categories (1) materialised systems and data warehouses, (2) systems that federate SPARQL engines and (3) live query approaches. Although our own work falls into the third category, in order to provide a broader background, in this section we also summarise developments in the first two categories. Additionally, we briefly remark on "hybrid" proposals that combine different methods, as well as proposals for new languages (aside from SPARQL) to query/navigate the Web of Data. Finally, since our extensions for LTBQE further relate to research on reasoning over Linked Data, we also cover some background in this area.

2.1. Materialised Systems

Materialised query-engines use a crawler or other data-acquisition methods to replicate (*i.e.*, *materalise*) remote Web of Data content into a local quad store over which queries are executed. Such services include "FactForge" [8][1] (which uses BigOWLIM [7]), "LOD Cache"[2] and Sindice's "Semantic Web Index" [45][3] (which both use Virtuoso [16]).

The typical goals of such materialised engines are to maintain broad and up-to-date coverage of the Web of Data and to process SPARQL queries efficiently. These objectives are (partially) met using distribution techniques, replication, optimised indexes, compression techniques, data synchronisation strategies, and so on [7,16,25,45]. Still, given that such services often index millions of documents, they require large amounts of resources to run. In particular, maintaining a local, optimal, *up-to-date* index with good coverage of the Web of Data is a Sisyphean task.

Unlike these materialised approaches, the LTBQE approach and our extensions do not require all content to be indexed locally prior to query execution.

[1]`http://factforge.net/sparql`
[2]`http://lod.openlinksw.com/sparql`
[3]`http://sparql.sindice.com/`

2.2. SPARQL Federation

Given the recent spread of independently operated SPARQL endpoints on the Web of Data hosting various closed datasets with varying degrees of overlap[4], *federated SPARQL querying* is enjoying growing attention in the research community. The core idea is to execute queries over a federation of endpoints: to split an input query, send the relevant sub-queries to individual (and possibly remote) endpoints *in situ*, and subsequently merge and process the final result set.

A primary challenge for federated SPARQL engines is to decide which parts of a query are best routed to which endpoint. Many such engines rely on "service descriptions" or "catalogues", which describe the contents of remote endpoints. One of the earliest such works (predating SPARQL by over three years) was by Stuckenschmidt *et al.* [54], who proposed summarising the content of distributed RDF repositories using schema paths (non-empty property chains). More recently, a variety of proposals have looked at using service descriptions or catalogues to enable routing in a federated scenario, including DARQ [49], SemWIQ [41], SPLENDID [20], ANAPSID [1], *etc.* Instead of relying (solely) on pre-computed service descriptions or catalogues, FedX [53] and SPLENDID (also) probe SPARQL endpoints with `ASK` sub-queries to test if they have relevant information.

New federation features were also introduced with SPARQL 1.1: the `SERVICE` feature invokes remote endpoints and the `VALUES` feature can be used to "ship" batches of intermediate bindings to an endpoint [23]. The SPARQL-DQP system [3] has investigated use of these SPARQL 1.1 federated features.

However, in recent work, we analysed a broad range of SPARQL endpoints available on the Web [4]. We found that half of the endpoints listed in the public `datahub.io` catalogue are no longer up, and that many endpoints have reliability issues. This work thus calls into question the current practicality of federation as a solution for querying the open Web.

Like the LTBQE approach and our proposed extensions, federated SPARQL engines may involve retrieving content from remote sources at runtime. However, unlike federated SPARQL, LTBQE operates over raw data sources on the level of HTTP, and does not require the availability of SPARQL interfaces.

[4]`http://www4.wiwiss.fu-berlin.de/lodcloud/state/`

2.3. Live Linked Data Querying

Live querying approaches access raw data sources at runtime to dynamically select, retrieve and build a dataset over which SPARQL queries can be evaluated. Ladwig & Tran [39] identify three categories of such query evaluation approaches: (1) top-down, (2) bottom-up, and (3) mixed strategy .

Top-down evaluation determines the query relevant sources before the actual query execution using knowledge about the available sources stored in a "source-selection index". These source-selection indexes can vary from simple inverted-index structures [42,45], to query-routing indexes [55], schema-level indexes [54], and lightweight hash-based structures [59].

The *bottom-up* query evaluation strategy involves discovering query-relevant sources on-the-fly during the evaluation of queries. The LTBQE approach [11, 24,28–30] we extend herein falls into this category (we define LTBQE in Section 4). The unique challenges for such an approach are (1) to find as many query-relevant sources as possible to improve recall of answers; (2) to conversely minimise the amount of sources accessed to avoid traffic and slow query-response times; (3) to optimise query execution in the absence of typical selectivity estimates, *etc.* [27,29]. In this paper, we focus on the first two challenges.

As its name suggests, the third strategy, the *mixed approach*, combines top-down and bottom-up techniques. This strategy uses (in a top-down fashion) some knowledge about sources to map query terms or query sub-goals to sources which can contribute answers, then discovering additional query relevant sources using a bottom-up approach [39].

2.4. Hybrid and Navigational Query Engines

A mature Linked Data query service could require a *hybrid strategy* that combines complementary approaches: for example to use a materialised approach for static datasets, a federated approach for dynamic datasets where a first-party SPARQL endpoint is available, or a live approach for dynamic datasets that have no SPARQL endpoint [38]. A number of works have investigated such combinations on a variety of levels (see, *e.g.*, [32, 40, 60]). In previous works we combined LTBQE with a materialised approach for getting fresher SPARQL query answers from the Web than the materialised indexes could offer but at a fraction of the runtime of pure LTBQE [60]. We believe that live query approaches, such as LTBQE, are most useful in combination with other query paradigms.

Some authors have also questioned whether or not SPARQL is the only language needed to query the Web of Data [5]. There have also been a number of proposals to extend SPARQL with regular expressions that capture navigational patterns[5], including work by Alkhateeb *et al.* [2], Perez *et al.*'s nSPARQL language [47], and Fionda *et al.*'s NautiLOD proposal [17]. Such work goes beyond pure SPARQL querying, but perhaps touches upon some of the broader potential of querying and consuming the Web of Data in a declarative manner.

2.5. Reasoning over Web Data

In this paper, we propose lightweight reasoning extensions for LTBQE, which leverage (some of) the semantics of the RDFS and OWL standards to perform inferencing and find additional answer and query relevant sources. We now cover some related works in the area of RDFS/OWL reasoning over the Web of Data.

We investigate a terse profile of reasoning targeted for Web data. Similarly, Glimm *et al.* [19] surveyed the use of RDFS and OWL features in a large crawl of the Web of Data (*viz.*, BTC'11), applying PageRank over documents and summating the rank of all documents using each feature. They found that RDF(S) features were the most prominently used. From OWL, `owl:sameAs` occurred most frequently, though features like `owl:FunctionalProperty` had higher ranks.

With respect to scalable rule-based reasoning over RDFS (and OWL), a number of authors have proposed separating schema data (aka. terminological data or T-Box) from instance data (aka. assertional data or A-Box) during inferencing [35, 61, 62]. The core assumptions are that the volume of schema data is much smaller than instance data, that schema data are frequently accessed during reasoning, that schema data are more static than instance data, and that schema-level inferences do not depend on instance data. Where these assumptions hold, the schema data can be separated from the main body of data and "compiled" into an optimised form in preparation for reasoning over the bulk of instance data. We use similar techniques herein when computing RDFS inferences: we consider schema data separately from instance data.

[5]SPARQL 1.1 includes a similar notion called *property paths* [23].

Aside from scalability, the freedom of the Web—where anyone can say anything (almost) anywhere—raises concerns about the trustworthiness of data for automated inferencing. On a schema level, for example, various obscure documents on the Web of Data make nonsensical definitions that would (naïvely) affect reasoning across all other documents [10]. Numerous authors have proposed mechanisms to incorporate notions of provenance for schema data into the inferencing process. One such procedure, called *authoritative reasoning*, only considers the schema definitions for a class or property term that are given in its respectively dereferenceable document [10,12,35]. We later use authoritative reasoning to avoid the unwanted effects of third-party schema contributions during RDFS reasoning. Delbru *et al.* [14] propose another solution called *context-dependent reasoning* (or quarantined reasoning), where a closed scope is defined for each document being reasoned over, incorporating only the document itself and other documents it (recursively) imports or links, thus excluding the claims of arbitrary Web documents. We later use a similar import mechanism to dynamically collect schemata from the Web during RDFS reasoning.

Our extensions involving `owl:sameAs` semantics do not directly involve schema data, but rather look at resolving coreferent resources in the corpus. Various authors have looked specifically at the use and the quality of use of `owl:sameAs` on the Web of Data [15,22,36]. Halpin *et al.* [22] look at the semantics and quality of `owl:sameAs` links in Linked Data. Manually inspecting five hundred `owl:sameAs` relations sampled from the Web of Data, they estimated an accuracy for `owl:sameAs` links—where sameness could be confidently asserted for the sampled relations—at around 51% (±21%), but found that judges often disagreed. We later conducted a similar manual inspection of one thousand `owl:sameAs` relations, where we asked a different question—is there any difference between these two resources to confirm that they are not the same?—and where we estimated a respective (and much more optimistic) precision of 97.2% [36]. We do not tackle issues pertaining to the quality of `owl:sameAs` relations in this particular work, but acknowledge this as an orthogonal challenge for Linked Data [22,36].

Finally, we are not the first work to look at incorporating reasoning into SPARQL querying over the Web of Data. Various materialised approaches for querying Linked Data have incorporated forward-chaining rule-based reasoners to find additional answers, including the SAOR reasoner [35] for YARS2 (which we use later), Sindice's context-dependent reasoning [14], Factforge [8], *etc.* In terms of reasoning for top-down querying systems, Li and Heflin [42] also use reasoning techniques to find additional results through query rewriting and bottom-up inferencing.

2.6. Novelty of Present Work

We briefly highlight our novelty. First and foremost, we evaluate the bottom-up LTBQE approach using various benchmarks—all in an uncontrolled, real-world setting—to see what kinds of practical expectations one can have for answering queries directly over the Web of Data. Second, we propose and evaluate reasoning extensions for LTBQE to find additional sources *on-the-fly* and to generate further answers. To the best of our knowledge, no other work has looked at evaluating live querying approaches over diverse sources live on the Web, nor has any other work looked at the benefits of incorporating reasoning techniques into link-traversal querying techniques for the Web of Data.

3. Preliminaries

In this section, we cover some necessary preliminaries and notation relating to RDF (§ 3.2), Linked Data (§ 3.3), SPARQL (§ 3.4) and RDFS & OWL (§ 3.5). Before we continue, however, we introduce a running example used to explain later concepts.

3.1. Running Example

Figure 1 illustrates an RDF (sub-)graph taken from five real-world interlinked documents on the Web of Data.[6] Prefixes for the abbreviated CURIE names used in this section, and throughout the paper, are available in Appendix C. The graph models information about two real-world persons and a paper that they coauthored together. One author is identified by the URIs `oh:olaf` and `dblpA:Olaf_Hartig` and the other by `cb:chris` and `dblpA:Christian_Bizer`. The URI `dblpP:HartigBF09` refers to the publication both authors share. The five documents are as follows:

`ohDoc:`, `cbDoc:` refer to the personal FOAF profile documents that each author created for themselves;

[6] As last accessed on 2013-07-02.

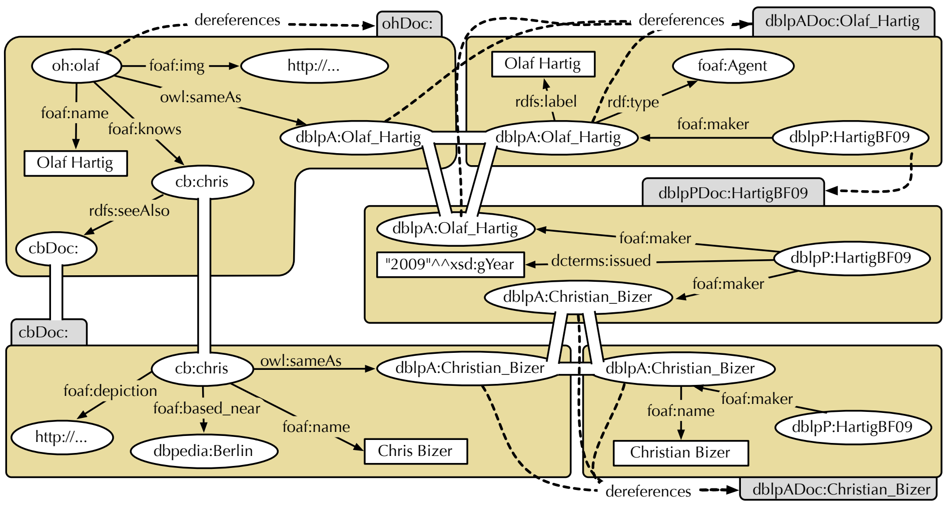


Fig. 1. Snippets taken from five documents on the Web of Data. Individual documents are associated with individual background panes. The URI of each document is attached to its pane with a shaded tab. The same resources appearing in different documents are joined using "bridges". Links from URIs to the documents they dereference to are denoted with dashed links. RDF triples are denoted following usual conventions within their respective document.

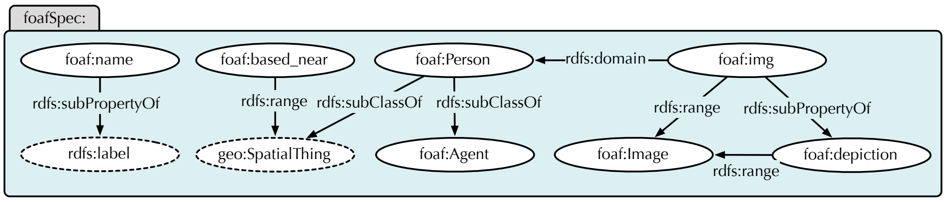


Fig. 2. Snippet from the Friend Of A Friend (FOAF) Ontology: a schema document on the Web of Data. External terms have dashed lines.

`dblpADoc:Olaf...`, `dblpADoc:Chris...` refer to information exported from the "DBLP Computer Science Bibliography"[7] for each author, including a publication list;

`dblpPDoc:HartigBF09` provides information about the co-authored paper exported from DBLP.

Each document is available as RDF/XML on the Web. Dereferenceable relationships between resources and documents are highlighted in Figure 1. Excluding `cbDoc:` (which must be looked up directly), the other four documents can be retrieved by dereferencing the URI of their main resource; for example, dereferencing `oh:olaf` over HTTP returns the document `ohDoc:` describing said resource.

In addition, Figure 2 illustrates a subset of RDFS definitions in a "schema document" extracted from the real-world FOAF ontology. Although left implicit, all terms in the `foaf:` namespace (including the predicates and values for `rdf:type` represented in Figure 1) dereference to this document. The relations between classes and properties shown in this document are well defined (using model-theoretic semantics) by the RDFS standard and can be used for automated inference [33].

[7]`http://www.informatik.uni-trier.de/~ley/db/`

3.2. RDF

In order to define our methods later, we must first provide notation for RDF [33].

Definition 1 (RDF Term, Triple and Graph)**.** *The set of* RDF terms *consists of the set of URIs* $\mathbf{U}$*, the blank-nodes* $\mathbf{B}$ *and literals* $\mathbf{L}$*. An* RDF triple $t := (s, p, o)$ *is an element of the set* $\mathbf{G} := \mathbf{UB} \times \mathbf{U} \times \mathbf{UBL}$ *(where, e.g.,* $\mathbf{UB}$ *is a shortcut for* $\mathbf{U} \cup \mathbf{B}$*). Here* s *is called subject,* p *predicate, and* o *object. An* RDF graph $G \subset \mathbf{G}$ *is a finite set of RDF triples. We use the* *functions* $\mathsf{subj}(G)$*,* $\mathsf{pred}(G)$*,* $\mathsf{obj}(G)$*,* $\mathsf{terms}(G)$ *to resp. denote the set of all terms appearing in the subject, predicate, object and all positions of triples in G.*

3.3. Linked Data

The four Linked Data principles [6] are as follows:

LDP1: *URIs are used to identify things*
LDP2: *URIs should be dereferenceable through HTTP*
LDP3: *Useful RDF content should be provided when URIs are dereferenced*
LDP4: *Links should be offered within that content*

We now provide some (time-invariant) Linked Data notation that we use later for defining our methods. This notation encapsulates the idea of the Web of Data as a database, where dereferencing URIs acts as a lookup that returns RDF data about those URIs.

Definition 2 (Data Source and Linked Dataset)**.** *We define the* http-download *function* $\mathsf{get} : \mathbf{U} \to 2^{\mathbf{G}}$ *as the mapping from URIs to RDF graphs provided by means of HTTP lookups which directly return status code* `200 OK` *and data in a suitable RDF format. We define the set of (RDF)* data sources $\mathbf{S} \subset \mathbf{U}$ *as the set of URIs* $\mathbf{S} := \{s \in \mathbf{U} : \mathsf{get}(s) \neq \emptyset\}$ *(i.e., URIs that return RDF content with* `200 Okay`*). We define a Linked Dataset as* $\Gamma \subset \mathsf{get}$ *(i.e., a finite set of pairs* $(s, \mathsf{get}(s))$ *such that* $s \in \mathbf{S}$*). The "global" RDF graph presented by a Linked Dataset is denoted as*

$$\mathsf{merge}(\Gamma) := \biguplus_{(u,G)\in\Gamma} G$$

where the operator '⊎' denotes the RDF merge of RDF graphs, which ensures that blank node labels in the input graphs are kept distinct in the merged graph [33].

Example 1. Taking Figure 1, *e.g.*, $\mathsf{get}(\texttt{ohDoc:}) = \{(\texttt{oh:olaf:}, \texttt{foaf:name}, \texttt{"Olaf Hartig"}), \ldots\}$, an RDF graph containing the five triples in that document. However, $\mathsf{get}(\texttt{oh:olaf}) = \emptyset$ since it does not return a `200 Okay` (redirects are supported in the next step). Thus, $\texttt{ohDoc:} \in \mathbf{S}$ whereas $\texttt{oh:olaf} \notin \mathbf{S}$. If we denote Figure 1 as the Linked Dataset Γ, we can say that $\Gamma = \{(\texttt{ohDoc:}, \mathsf{get}(\texttt{ohDoc:}), \ldots\}$, containing five (*URI*, *RDF-graph*) pairs. Then, $\mathsf{merge}(\Gamma)$ is the set of all 17 RDF triples shown in Figure 1. □

Definition 3 (Dereferencing RDF)**.** *A URI may redirect to another URI with a* `30x` *response code. We denote this function as* $\mathsf{redir} : \mathbf{U} \to \mathbf{U}$*, which first strips the fragment identifier of a URI (if present) and would then map a URI to its redirect target or to itself in the case of failure (e.g., where no redirect exists). We denote the fixpoint of* redir *as* redirs*, denoting traversal of a number of redirects (a limit may be imposed to avoid cycles). We then denote* dereferencing *by the composition* $\mathsf{deref} := \mathsf{get} \circ \mathsf{redirs}$*, which maps a URI to an RDF graph retrieved with status code* `200 OK` after *following redirects, or which maps a URI to the empty set in the case of failure. We denote the set of* dereferenceable URIs *as* $\mathbf{D} := \{d \in \mathbf{U} : \mathsf{deref}(d) \neq \emptyset\}$*, where* $\mathbf{S} \subset \mathbf{D}$ *and we place no expectations on what* $\mathsf{deref}(d)$ *returns, other than returning some valid RDF. As a shortcut, we denote by* $\mathsf{derefs} : 2^{\mathbf{U}} \to 2^{\mathbf{U}\times 2^{\mathbf{G}}}; U \mapsto \{(\mathsf{redirs}(u), \mathsf{deref}(u)) \mid u \in U \cap \mathbf{D})\}$ *the mapping from a set of URIs to the Linked Dataset it represents by dereferencing all URIs (only including those in* $\mathbf{D}$ *which return some RDF).*

In relation to the formal model of Hartig [28], we favour concrete HTTP-level methods used for Linked Data. He models the Web of Linked Data as a triple $W = (D, data, adoc)$, where our set $\mathbf{S}$ is equivalent to his set D of document IDs, our function $\mathsf{get}(.)$ instantiates his (more general) function $data(.)$ for mapping document IDs to RDF graphs, and our function $\mathsf{redirs}(.)$ instantiates his function $adoc(.)$ for partially mapping URI names to document IDs.

Example 2. Taking Figure 1, `oh:olaf` redirects to `ohDoc:`, denoted $\mathsf{redir}(\texttt{oh:olaf}) = \texttt{ohDoc:}$. No further redirects are possible, and thus $\mathsf{redirs}(\texttt{oh:olaf}) = \texttt{ohDoc:}$. Dereferencing `oh:olaf` gives the RDF graph in the document `ohDoc:`, where $\mathsf{deref}(\texttt{oh:olaf}) = \mathsf{get}(\mathsf{redirs}(\texttt{oh:olaf})) = \mathsf{get}(\texttt{ohDoc:})$. Instead taking the URI `cb:chris`, $\mathsf{redir}(\texttt{cb:chris}) = \texttt{cb:chris}$ and $\mathsf{get}(\texttt{cb:chris}) = \emptyset$; this URI is not dereferenceable. Thus we can say that $\texttt{oh:olaf} \in \mathbf{D}$ and $\texttt{ohDoc:} \in \mathbf{D}$ whereas $\texttt{cb:chris} \notin \mathbf{D}$. □

3.4. SPARQL

We now introduce some concepts relating to the query language SPARQL [46, 48]. We herein focus on evaluating simple, conjunctive, *basic graph patterns* (BGPs), where although supported by our implementation, we do not formally consider more expressive parts of the SPARQL language, which—with the exception of non-monotonic features that assume a closed dataset like `OPTIONAL` in SPARQL and `MINUS`/(`NOT`) `EXISTS` in SPARQL 1.1 —can be layered on top [46]. In addition, we consider URIs and not IRIs for convenience with the RDF preliminaries.

Definition 4 (Variables, Triple Patterns & BGPs). *Let $\mathbf{V}$ be the set of variables ranging over $\mathbf{UBL}$. A* triple pattern $tp := (s, p, o)$ *is an element of the set* $\mathbf{Q} := \mathbf{VUL} \times \mathbf{VU} \times \mathbf{VUL}$.[8] *For simplicity, we do not consider blank-nodes in triple patterns (they could be roughly emulated by an injective mapping from $\mathbf{B}$ to $\mathbf{V}$). A finite (herein, non-empty) set of triple patterns $Q \subset \mathbf{Q}$ is called a* Basic Graph Pattern*, or herein, simply a* query*. We use* $\mathsf{vars}(Q) \subset \mathbf{V}$ *to denote the set of variables in Q. Finally, we may overload graph notation for queries, where, e.g.,* $\mathsf{terms}(Q)$ *returns all elements of* $\mathbf{VUL}$ *in Q.*

In this paper, we only look at evaluating queries representing BGPs. We denote the answers to a query Q with respect to a Linked Dataset as $[\![Q]\!]_\Gamma$. We now formally define this notion, where $[\![Q]\!]_\Gamma$ is a set of solutions generated by Q over the merge of graphs in Γ.

Definition 5 (SPARQL solutions). *Call the partial function $\mu : \mathrm{dom}(\mu) \cup \mathbf{UL} \to \mathbf{UBL}$ a* solution mapping *with a domain $\mathrm{dom}(\mu) \subset \mathbf{V}$. A solution mapping binds variables in $\mathrm{dom}(\mu)$ to $\mathbf{UBL}$ and is the identify function for $\mathbf{UL}$. Overloading notation, let $\mu : \mathbf{Q} \to \mathbf{G}$ and $\mu : 2^{\mathbf{Q}} \to 2^{\mathbf{G}}$ also resp. denote a solution mapping from triple patterns to RDF triples, and basic graph patterns to RDF graphs such that $\mu(tp) := (\mu(s), \mu(p), \mu(o))$ and $\mu(Q) := \{\mu(tp) \mid tp \in Q\}$. We now define the set of* SPARQL solutions *for a query Q over a (Linked) Dataset Γ as*

$$[\![Q]\!]_\Gamma := \{\mu \mid \mu(Q) \subseteq \mathsf{merge}(\Gamma) \wedge \mathrm{dom}(\mu) = \mathsf{vars}(Q)\} .$$

For brevity, and unlike SPARQL, solutions are herein given as sets (not multi-sets), implying a default `DISTINCT` semantics for queries, and we assume that answers are given over the default graph consisting of the merge of RDF graphs in the dataset.

Example 3. Taking Γ from Figure 1, let Q be:

```
SELECT ?maker ?issued WHERE {
  dblpP:HartigBF09 foaf:maker ?maker ;
     dcterms:issued ?issued . }
```

Then $[\![Q]\!]_\Gamma$ would be:

?maker	?issued
dblpA:Christian_Bizer	"2009"^^xsd:gYear
dblpA:Olaf_Hartig	"2009"^^xsd:gYear

□

3.5. RDFS and OWL

In preparation for defining our reasoning extensions to LTBQE, we now give some preliminaries relating to RDFS and OWL. Our RDFS rules are the subset of the ρDF rules proposed by Muñoz *et al.* [44] that deal with instance data entailments (as opposed to schema-level entailments).[9] For supporting `owl:sameAs`, we use a small subset of OWL 2 RL/RDF rules, given in Table 1, which constitute a partial axiomatisation of the OWL RDF-Based Semantics relating to ground equality. Our selection of rules thus support a small subset of standard RDFS/OWL semantics that we argue to be important for answering queries over the Web of Data, and our approach could be generalised to support, *e.g.*, OWL 2 RL/RDF rules with some adaptations.

For a ruleset R and dataset Γ, we denote by $\Gamma \bullet R$ the fixpoint of applying R against Γ such that applying R over $\Gamma \bullet R$ yields no new inferences. We now define $\Gamma \bullet R$ formally in terms of an immediate consequence operator (reusing some SPARQL notation for brevity).

Definition 6 (Entailment Rules & Least Model). *An entailment rule is a pair $r = (Body, Head)$ (cf. Table 1) such that $Body, Head \subset \mathbf{Q}$; and* $\mathsf{vars}(Head) \subseteq \mathsf{vars}(Body)$*. The immediate consequences of r for a Linked Dataset Γ are given as:*

$$\mathfrak{T}_r(\Gamma) := \{\mu(Head) \mid \mu \in [\![Body]\!]_\Gamma\} \setminus \mathsf{merge}(\Gamma) .$$

In other words, $\mathfrak{T}_r(\Gamma)$ denotes the direct unique inferences from a single application of a rule r against the merge of RDF data contained in Γ. Let R denote a finite set of entailment rules. The immediate consequences of R over Γ are given analogously as:

$$\mathfrak{T}_R(\Gamma) := \textstyle\bigcup_{r \in R} \mathfrak{T}_r(\Gamma) .$$

This is the union of a single application of all rules in R over the data applied to the (raw) data in Γ. Furthermore, let $\delta \in \mathbf{U}$ denote a fresh URI which names the graph G^R of data inferred by R, and let $G^R_0 = \emptyset$. Now, for $i \in \mathbb{N}$, define:

$$\Gamma^R_i := \Gamma \cup \big\{\big(\delta, G^R_i\big)\big\}$$

$$G^R_{i+1} := \mathfrak{T}_R(\Gamma^R_i) \cup G^R_i$$

The least model *of Γ with respect to R is Γ^R_n for the least n such that $\Gamma^R_n = \Gamma^R_{n+1}$; at this stage the closure*

[8]SPARQL allows literals in the subject position [48], though such patterns cannot match RDF triples as currently defined.

[9]We drop *implicit typing* [44] rules as we allow generalised RDF in intermediate inferences.

	ID	Body	Head
RDFS	PRP-SPO1	`?p₁ rdfs:subPropertyOf ?p₂ . ?s ?p₁ ?o .`	`?s ?p₂ ?o .`
	PRP-DOM	`?p rdfs:domain ?c . ?s ?p ?o .`	`?p a ?c .`
	PRP-RNG	`?p rdfs:range ?c . ?s ?p ?o .`	`?o a ?c .`
	CAX-SCO	`?c₁ rdfs:subClassOf ?c₂ . ?s a ?c₁ .`	`?s a ?c₂ .`
Same-As	EQ-SYM	`?x owl:sameAs ?y .`	`?y owl:sameAs ?x .`
	EQ-TRANS	`?x owl:sameAs ?y . ?y owl:sameAs ?z .`	`?x owl:sameAs ?z .`
	EQ-REP-S	`?s owl:sameAs ?s′ . ?s ?p ?o .`	`?s′ ?p ?o .`
	EQ-REP-P	`?p owl:sameAs ?p′ . ?s ?p ?o .`	`?s ?p′ ?o .`
	EQ-REP-O	`?o owl:sameAs ?o′ . ?s ?p ?o .`	`?s ?p ?o′ .`

Table 1
RDFS (ρDF subset) and `owl:sameAs` (OWL 2 RL/RDF subset) rules

is reached and nothing new can be inferred.[10] *Henceforth, we denote this least model with* $\Gamma \bullet R$. *Query answers including entailments are given by* $[\![Q]\!]_{\Gamma \bullet R}$.

Example 4. Let R denote the set of rules in Table 1. Also, consider Γ as the Linked Dataset comprising of $(\texttt{ohDoc:}, \mathsf{get}(\texttt{ohDoc:}))$ from Figure 1 and a second named graph called `foafSpec:` with the following subset of triples from Figure 2:

```
foaf:img rdfs:domain foaf:Person ;
        rdfs:range foaf:Image ;
        rdfs:subPropertyOf foaf:depiction .
foaf:Person rdfs:subClassOf foaf:Agent .
```

These (real-world) triples can be retrieved by dereferencing a FOAF term; *e.g.*, $\mathsf{deref}(\texttt{foaf:img})$. Now, given Γ and R, then $G_0^R = \emptyset$, $G_1^R = G_0^R \cup \mathfrak{T}_R(\Gamma_0^R)$ where, by applying each rule in R over Γ once, $\mathfrak{T}_R(\Gamma_0^R)$ contains the following triples (abbreviating CURIEs slightly):

```
oh:olaf foaf:depiction <http...> .        #PRP-SPO1
oh:olaf a foaf:Person .                   #PRP-DOM
<http...> a foaf:Image .                  #PRP-RNG
dblpA:Olaf owl:sameAs oh:olaf .           #EQ-SYM
dblpA:Olaf foaf:knows cb:chris .          #EQ-REP-S
...
```

Subsequently, $\Gamma_1^R = \Gamma \cup \{(\delta, G_1^R)\}$, where δ is any built-in URI used to identify the graph of inferences and where G_1^R contains the unique inferences thus far (listed above). Thereafter, $G_2^R = G_1^R \cup \mathfrak{T}_R(\Gamma_1^R)$, where $\mathfrak{T}_R(\Gamma_1^R)$ contains:

```
oh:olaf a foaf:Agent .                         #CAX-SCO
dblpA:Olaf foaf:depiction <http...> .          #EQ-REP-S
dblpA:Olaf a foaf:Person .                     #EQ-REP-S
dblpA:Olaf owl:sameAs dblpA:Olaf .             #EQ-REP-S
oh:olaf owl:sameAs oh:olaf .                   #EQ-REP-S
...
```

As before, $\Gamma_2^R = \Gamma \cup \{(\delta, G_2^R)\}$, where G_2^R contains all inferences collected thus far, and $G_3^R = G_2^R \cup \mathfrak{T}_R(\Gamma_2^R)$, where $\mathfrak{T}_R(\Gamma_2^R)$ contains:

```
dblpA:Olaf a foaf:Agent .                      #CAX-SCO
```

This is then the closure since $\mathfrak{T}_R(\Gamma_3^R) = \emptyset$; nothing new can be inferred, and so $\Gamma_3^R = \Gamma_4^R$. And thus we can say that $\Gamma \bullet R = \Gamma_3^R = \Gamma \cup (\delta, G_3^R)$. $\square$

4. Link Traversal Based Query Execution

Having covered some necessary preliminaries, in the following section, we cover the Link Traversal Based Query Execution (LTBQE) approach first proposed by Hartig *et al.* [28, 29] for executing SPARQL queries over the Web of Data (§ 4.1).

4.1. Overview of Baseline LTBQE

Given a SPARQL query, the core operation of LTBQE is to identify and retrieve a focused set of query-relevant RDF documents from the Web of Data from which answers can be extracted. The approach begins by dereferencing URIs found in the query itself. The documents that are returned are parsed, and triples matching patterns of the query are processed; the URIs in these triples are also dereferenced to look for further information, and so forth. The process is recursive

[10]Since our rules are a syntactic subset of Datalog, there is a unique and finite least model (assuming finite inputs).

up to a fixpoint wherein no new query-relevant sources are found. New answers for the query can be computed on-the-fly as new sources arrive. We now formally define an idea of query-relevant documents in the context of LTBQE. This is similar in principle to the generic notion of reachability introduced previously [28, 30], but relies here on concrete HTTP specific operations:

Definition 7 (Query Relevant Sources & Answers)**.** *First let* $\mathsf{uris}(\mu) := \{u \in \mathbf{U} \mid \exists v \text{ s.t. } (v,u) \in \mu\}$ *denote the set of URIs in a solution mapping* μ*. Given a query* Q *and an intermediate dataset* Γ*, we define the function* qrel*, which extracts from* Γ *a set of URIs that can (potentially) be dereferenced to find further sources deemed relevant for* Q*:*

$$\mathsf{qrel}(Q,\Gamma) := \bigcup_{tp \in Q} \bigcup_{\mu \in [\![\{tp\}]\!]_\Gamma} \mathsf{uris}(\mu)$$

To begin the recursive process of finding query-relevant sources, LTBQE takes URIs in the query—denoted with $U_Q := \mathsf{terms}(Q) \cap \mathbf{U}$*—as "seeds", and builds an initial dataset by dereferencing these URIs:* $\Gamma_0^Q := \mathsf{derefs}(U_Q)$*. Thereafter, for* $i \in \mathbb{N}$*, define:*[11]

$$\Gamma_{i+1}^Q := \mathsf{derefs}\big(\mathsf{qrel}(Q, \Gamma_i^Q)\big) \cup \Gamma_i^Q$$

The set of LTBQE query relevant sources *for* Q *is given as the least* n *such that* $\Gamma_n^Q = \Gamma_{n+1}^Q$*, denoted simply* Γ^Q*. The set of* LTBQE query answers *for* Q *is given as* $[\![Q]\!]_{\Gamma^Q}$*, or simply denoted* $[\![Q]\!]$*.*

Example 5. Taking Figure 1, let Q be the following query looking for the author-names of a given paper:

```
SELECT ?authorName WHERE {
  dblpP:HartigBF09 foaf:maker ?author .
  ?author foaf:name ?authorName . }
```

First, the process extracts all raw query URIs: $U_Q = \{$`dblpP:HartigBF09`, `foaf:name`, `foaf:maker`$\}$. In the next stage, the engine dereferences these URIs. Given that redirs(`dblpP:HartigBF09`) = `dblpPDoc:HartigBF09` & redirs(`foaf:maker`) = redirs(`foaf:made`) = `foafSpec:`, dereferencing U_Q gives two unique named graphs, *viz.*: (`dblpPDoc:HartigBF09`, get(`dblpPDoc:HartigBF09`)) and (`foafSpec:`, get(`foafSpec:`)). These two named-graphs comprise Γ_0^Q. (In fact, only the former graph will ultimately contribute answers.)

Second, LTBQE looks to extract additional query relevant URIs by seeing if any query patterns are matched in the current dataset. By reference to the graph `dblpPDoc:HartigBF09` in Figure 1, we see that for the pattern "`dblpP:HartigBF09 foaf:maker ?author .`", the variable `?author` is matched by two unique URIs, namely `dblpA:Christian_Bizer` and `dblpA:Olaf_Hartig`, which are added to $\mathsf{qrel}(Q, \Gamma_0^Q)$. Nothing else is matched. Hence, these two URIs are dereferenced and the results added to Γ_0^Q to form Γ_1^Q.

LTBQE repeats the above process until no new sources are found. At the current stage, Γ_1^Q now also contains the two sources `dblpADoc:Christian_Bizer` and `dblpADoc:Olaf_Hartig` needed to return:

?authorName
"Christian Bizer"
"Olaf Hartig"

Furthermore, no other query-relevant URIs are found and so a fixpoint is reached and the process terminates: $[\![Q]\!]$ contains the above results. □

4.2. Decidability and completeness

The decidability of LTBQE—and indeed the decidability of the more general problem of evaluating a SPARQL query over the Web of Data—depends on how one scopes the sources of data considered for evaluation and which features of SPARQL are used.

If one considers an infinite Web of Data—aiming for what Hartig calls "*Web completeness*" [28]—then the evaluation of a SPARQL query is not finitely computable in the general case, even if one considers only the monotonic features of SPARQL [28]: there are (countably) infinite documents that may contain relevant data but that cannot be processed in finite steps. If non-monotonic features of SPARQL—such as `OPTIONAL`, `MINUS`, *etc.*—are used in the query, then evaluation is not even "*eventually computable*" by a non-terminating procedure [28] since, in the general case, all documents need to be processed before a single sound solution can be given, making the computation of each individual solution infinitary.

If one rather considers a reachability condition—whereby, for example, only the query-relevant sources in an LTBQE sense are considered in-scope—for similar reasons, Hartig [28] shows that queries are still not finitely computable unless it can be proven that the number of reachable sources is finite under the given

[11] In practice, URIs need only be dereferenced once; *i.e.*, only URIs in $\mathsf{qrel}(Q, \Gamma_i^Q) \setminus (\mathsf{qrel}(Q, \Gamma_{i-1}^Q) \cup U_Q)$ need be dereferenced at each stage.

conditions. This does not hold for the set of query relevant sources given in Definition 7.

Example 6. The following query asks for a general description of people known by `oh:olaf`:

```
SELECT ?s ?p ?o WHERE {
  oh:olaf foaf:knows ?s .
  ?s ?p ?o . }
```

The initial query-relevant sources (per Definition 7) are the documents dereferenced from `oh:olaf` and `foaf:maker`. Thereafter, all triples in these documents will match the open pattern, and thus all URIs in these documents will be considered as potential query-relevant links. This will continue recursively, crawling the entire, potentially infinite Web of Data as reachable from the query URIs. The problem is not limited to open patterns; take the following query:

```
SELECT ?o WHERE {
  oh:olaf foaf:knows ?s .
  ?s foaf:knows ?o . }
```

This would end up crawling the connected Web of FOAF documents, as are linked together by dereferenceable `foaf:knows` links. □

Partly addressing this problem, Hartig *et al.* [29] defined an iterator-based execution model for LTBQE, which rather approximates the answers provided by Definition 7. This execution model defines an ordering of triple patterns in the query, similar to standard nested-loop join evaluation. The most selective patterns (those expected to return the fewest bindings) are executed first and initial bindings are propagated to bindings further up the tree. Crucially, later triple patterns are partially bound when looking for query-relevant sources. Thus, taking the previous example, the pattern "`?s foaf:knows ?o .`" will never be used to find query-relevant sources, but rather partially-bound patterns like "`cb:chris foaf:knows ?o .`" will be used. As such, instead of retrieving all possible query-relevant sources, the iterator-based execution model uses interim results to apply a more focused traversal of the Web of Data. This also makes the iterator-based implementation order-dependent: results may vary depending on which patterns are executed first and thus answers may be missed. However, it does solve the problem of traversing too many sources when low-selectivity patterns are present in the query.

Relatedly, Harth and Speiser [24] also consider an order-dependent version of LTBQE. Similar to Hartig [29], they remark that although the *Web-completeness* of SPARQL evaluation is useful as a theoretical notion, since the Web of Data cannot be materialised, more practical but weaker notions of completeness are also necessary. They propose another two completeness conditions: *seed-complete* considers answers from the set of documents that are within a fixed-length traversal path from the seed URIs in the query (the authors propose using the number of query patterns to decide the maximum hops) and *query-reachable-complete* considers a closed dataset of documents reachable through the LTBQE process under some ordering of the query patterns. The authors then demonstrate how *Web-complete* and *query-reachable-complete* conditions coincide if only certain authoritative triples—triples containing URIs that dereference to the container document—are considered.

We now give some practical examples as to why LTBQE (be it order-dependent or order-independent) cannot be Web-complete in the general case.

No dereferenceable query URIs: The LTBQE approach cannot return results in cases where the query does not contain dereferenceable URIs. For example, consider posing the following query against Figure 1:

```
SELECT * WHERE {
  cb:chris ?p ?o . }
```

As previously explained, the URI `cb:chris` is not dereferenceable (deref(`cb:chris`) $= \emptyset$) so the query has no place to start its traversal from.

Unconnected query-relevant documents: Relating to reachability, results can be missed where documents are "connected" by a join on literals, blank-nodes or non-dereferenceable URIs. The following query illustrates such a case:

```
SELECT ?olaf ?name WHERE {
  oh:olaf foaf:name ?name .
  ?olaf foaf:name ?name . }
```

Answers (other than `oh:olaf`) cannot be reached from the starting URI `oh:olaf` because the relevant documents are connected by the literal `"Olaf Hartig"`.

Dereferencing partial information: In the general case, the effectiveness of LTBQE is heavily dependent on the amount of data returned by the deref(u) function. In an ideal case, dereferencing a URI u would return all triples mentioning u on the Web of Data. However, this is not always the case; for example:

```
SELECT ?s WHERE {
  ?s owl:sameAs dblpA:Olaf_Hartig . }
```

This simple query cannot be answered since the triple "`oh:olaf owl:sameAs dblpA:Olaf_Hartig .`" is not accessible by dereferencing `dblpA:Olaf_Hartig`. The assumption that all RDF available on the Web of Data about a URI u can be collected by dereferencing u is clearly idealised; hence, later in Section 6 we will empirically analyse how much the assumption holds in practice, giving insights into the potential recall of LTBQE on an infrastructural level.

5. LiDaQ: Extending LTBQE with Reasoning

We now present the details of LiDaQ: our proposal to extend the baseline LTBQE approach with components that leverage lightweight RDFS and `owl:sameAs` reasoning in order to improve recall. We first describe the extensions we propose (§ 5.1), and then describe our implementation of the system (§ 5.2).

5.1. LTBQE Extensions

Partly addressing some of the shortcomings of the LTBQE approach in terms of "recall", Hartig *et al.* [29] proposed extending the set of query relevant sources to consider `rdfs:seeAlso` links, which sometimes overcomes the issue of URIs not being dereferenceable. In the LiDaQ system, we include this extension and further propose novel extensions that apply reasoning over query-relevant sources to squeeze additional answers from these sources, which in turn may lead to recursively finding additional query-relevant sources.

We now describe, formally define and provide motivating examples for each of the three extensions: following `rdfs:seeAlso` links, following `owl:sameAs` links and applying equivalence inferencing, and collecting schema information for applying RDFS reasoning.

5.1.1. Following `rdfs:seeAlso` links:

We first motivate the legacy `rdfs:seeAlso` extension with a simple example.

Example 7. Consider executing the following simple query, asking for images of the friends of `oh:olaf`, against the data in Figure 1 using baseline LTBQE:

```
SELECT ?f ?d WHERE {
  oh:olaf foaf:knows ?f .
  ?f foaf:depiction ?d . }
```

LTBQE first dereferences the content of the query URI `oh:olaf`, and then follows and dereferences all URI bindings for the variable `?f`, matching the second query pattern "`?f foaf:depiction ?d .`" over the retrieved content to find pictures. However, the query processor needs to follow the `rdfs:seeAlso` link from `cb:chris` (bound to `?f`) to `cbDoc:` since the URI `cb:chris` is not dereferenceable (recall that a dashed arrow in Figure 1 denotes dereferenceability). □

Hartig *et al.* [29] thus proposed to extend LTBQE to consider `rdfs:seeAlso` links as follows.

Definition 8 (LTBQE Extension 1: `rdfs:seeAlso`). *Given a dataset Γ, a set of URIs U and a predicate URI p, first define:*

$$\mathsf{link}(\Gamma, U, p) := \{v \in \mathbf{U} \mid \exists u \in U \text{ s.t. } (u, p, v) \in \mathsf{merge}(\Gamma)\}$$

which gives the target of links from URIs in U with the property p in the data of Γ. Next we extend the $\mathsf{qrel}(\Gamma, U)$ function from Definition 7 to allow for considering links through a predicate p as follows:

$$\mathsf{qrel}(Q, \Gamma, p) := \mathsf{qrel}(Q, \Gamma) \cup \mathsf{link}\big(\Gamma, \mathsf{qrel}(Q, \Gamma), p\big)$$

Note that $\mathsf{qrel}(Q, \Gamma) \subseteq \mathsf{qrel}(Q, \Gamma, p)$. The extension to follow `rdfs:seeAlso` links then follows from Definition 7 by replacing $\mathsf{qrel}(\Gamma, U)$ with the extended function $\mathsf{qrel}(\Gamma, U,$ `rdfs:seeAlso`$)$.

5.1.2. Following and reasoning over `owl:sameAs` links:

We now motivate the need for `owl:sameAs` traversal and reasoning with an example:

Example 8. Consider the following query for Figure 1 asking for friends of `oh:olaf` that are also co-authors.

```
SELECT ?f WHERE {
  oh:olaf foaf:knows ?f , foaf:maker ?p .
  ?f foaf:maker ?p }
```

Baseline LTBQE returns no answers: LTBQE requires `owl:sameAs` support to return Chris as an answer (given by the equivalences for `oh:olaf`/`dblpA:Olaf_Hartig` and `cb:chris`/`dblpA:Christian_Bizer`). □

We now formalise the details of our extension.

Definition 9 (LTBQE Extension 2: `owl:sameAs`). *We define an extension of LTBQE to consider `owl:sameAs` links and inferences. As before, from Definition 7, replace* $\mathsf{qrel}(\Gamma, U)$ *with* $\mathsf{qrel}(\Gamma, U,$ `owl:sameAs`$)$*, which follows `owl:sameAs` links. Next, let* $R^{=}$ *denote the set of rules of the form* EQ-* *in Table 1. Finally, from Definition 7, replace* Γ_i^Q *with* $\Gamma_i^Q \bullet R^{=}$*, such that the `owl:sameAs` inferences are applied at each step.*

5.1.3. Incorporating RDFS schemata and reasoning

Finally, we cover our novel extension for RDFS inferencing, starting with a motivating example.

Example 9. Take the following query over Figure 1 asking for the images(s) depicting `oh:olaf`:

```
SELECT ?d WHERE {
  oh:olaf foaf:depiction ?d . }
```

From Figure 2, we know that `foaf:depiction` is a sub-property of `foaf:img`, and we would thus hope to get the answer `<http://...>` from `ohDoc:`. However, returning this answer requires two thing: (i) retrieving the RDFS definitions of the FOAF vocabulary; and (ii) performing reasoning using the first four rules in Table 1. In this case, finding the relevant schema information (the first step) is quite straightforward and can be done dynamically since the relevant terms (`foaf:img` and `foaf:depiction`) are within the same namespace and are described by the same dereferenceable document. However, consider instead:

```
SELECT ?d WHERE {
  oh:olaf rdfs:label ?d . }
```

In this case, we know from the FOAF schema that `foaf:name` is a sub-property of `rdfs:label`, and so `"Olaf Hartig"` should be an answer. However, no FOAF vocabulary term is mentioned in the query, and so the FOAF schema will not be in the query-relevant scope. To overcome this, we can provide a static set of schema information to the query engine as input, or we can dereference property and class terms mentioned in the query-relevant data to dynamically retrieve the relevant definitions at runtime. □

Definition 10 (LTBQE Extension 3: RDFS). *We define an extension of LTBQE to consider RDFS schema data and a subset of RDFS inferences. Let* R^{ρ} *denote rules* $\{$PRP-SPO1, PRP-DOM, PRP-RNG, CAX-SCO$\}$ *in Table 1 (other rules could be added as necessary). From Definition 7, replace* Γ_i^Q *with* $(\Gamma_i^Q \cup \Psi_i) \bullet R^{\rho}$*, such that inferences are applied at each step. We use* Ψ_i *to denote an auxiliary Linked Dataset containing schema data at step* i*.*

We are then left to define how Ψ_i *may be acquired, where we provide three options (*Ψ_i^{a-c}*).*

1. *A static corpus of schema data* Ψ *can be provided as input, such that* $\Psi_i^a := \Psi$*.*
2. *The class and property terms used in* Γ_i^Q *can be dereferenced. Letting* $\mathsf{preds}(\Gamma)$ *and* $\mathsf{o\text{-}type}(\Gamma)$ *denote, respectively, the set of all URIs appearing as a predicate, and the set of all URIs appearing as a value for `rdf:type` (class instance) in* Γ*, we can define* Ψ_i^b *as:*

$$\Psi_i^b := \mathsf{derefs}\big(\mathsf{preds}(\Gamma_i^Q) \cup \mathsf{o\text{-}type}(\Gamma_i^Q)\big)$$

3. *Class and property terms can be dereferenced and schema-level links followed. For a Linked Dataset* Γ*, let* $\mathsf{imports}(\Gamma)$ *denote all URIs appearing as the subject or object of a triple in* Γ *with predicate `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range` or `rdfs:subClassOf`; or appearing as the object of `owl:imports`. Extending* Ψ_i^b *as above, let* $\Psi_{i,0}^c := \Psi_i^b$*, and thereafter, for* $j \in \mathbb{N}$ *define:*

$$\Psi_{i,j+1}^c := \mathsf{derefs}\big(\mathsf{imports}(\Psi_{i,j}^c)\big) \cup \Psi_{i,j}^c$$

such that links are recursively followed up to a fixpoint: the least j *such that* $\Psi_{i,j}^c = \Psi_{i,j+1}^c$*. We define* Ψ_i^c *as this fixpoint of recursive imports.*

The second and third methods involve dynamically collecting schemata at runtime. The third method of schema-collection is potentially problematic in that it recursively follows links, and may end up collecting a large amount of schema documents (a behaviour we encounter in evaluation later). However, where, for example, class or property hierarchies are split across multiple schema documents, this recursive process is required to “recreate” the full hierarchy.

All three extensions—following `rdfs:seeAlso` links, following `owl:sameAs` links & applying `owl:sameAs` reasoning, retrieving RDFS data (using one of three approaches) & applying RDFS reasoning—can be combined in a straightforward manner. In fact, some answers may only be possible through the combination of all extensions. We will later explore the effects of combining all extensions in Section 8.

5.2. LiDaQ Implementation

The LiDaQ prototype (implemented in Java) draws together a variety of techniques proposed in the literature [29, 31, 40] and has five main components, as depicted in Figure 3.

***Query Processor*:** uses Jena ARQ to parse and process input SPARQL queries and format the output results.[12] We discuss query processing in further detail below.

***Source Selector*:** decides which query and solution URIs should be dereferenced and which links should be followed.

***Source Lookup*:** an adapted version of the LDSpider crawling framework performs the live Linked Data lookups required for LTBQE. LDSpider respects the `robots.txt` policy, blacklists typical non-RDF URI patterns (*e.g.,* `.jpeg`) and enforces a half-second politeness delay between two consequential lookups for URIs hosted at the same pay-level-domain. A per-domain queue is implemented from which a pool of threads polls on a first-in-first-out basis.[13]

***Local Repository*:** a custom implementation of an in-memory quad store (similar to [31]) is used to cache the content of all query relevant data (including inferences and schema data), as well as indexing triple patterns from the query to match against the data. Triple-pattern "listeners" match cached data in a continuous fashion, feeding the iterators.

***Reasoner*:** the Java-based SAOR reasoner is used to support rule-based reasoning extensions [10] and executes inferencing over the local repository.

The query processing algorithm is based on a nested-loop strategy. During LTBQE, retrieving certain sources may involve high latency. Thus rather than blocking while waiting for the result of a particular request, special non-blocking operators are required [29, 40], where we adopt a strategy analogous to the *symmetric-index-hash join* [40]. When a data-access operator receives a lookup request, it: (i) registers the pattern in a hash-table, (ii) pulls all relevant data, currently found in the local repository, as bindings into the hash-table, (iii) co-ordinates with the source-selector to request remote accesses as necessary, and (iv) registers a listener with the local repository that will push new relevant data into the operator's hash-table as they arrive. Join operators use these hash-tables to asynchronously concatenate compatible tuples and return them to higher operators. By caching intermediate bindings, this asynchronous model ensures that all relevant data are consumed—in a bottom-up, nested-loop sense—as they arrive, no matter when or from where they arrive [40].

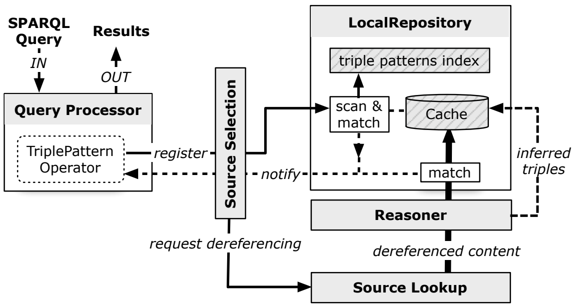


Fig. 3. LTBQE architecture diagram.

We further investigate some practical optimisations to minimise the number of query-relevant sources retrieved while maximising results. First, we avoid dereferencing URIs that do not appear in join positions. We illustrate this with a simple example:

Example 10. Consider the following query issued against the example graph of Figure 1, asking for friends of `oh:olaf` that have some value defined for `foaf:based_near`:

```
SELECT ?f ?fn ?b WHERE {
   oh:olaf foaf:knows ?f .
   ?f foaf:name ?fn .
   ?f foaf:based_near ?b . }
```

Assuming the `rdfs:seeAlso` extension is enabled, LTBQE will visit `ohDoc`: binding `cb:chris` for `?f`; and then visit `cbDoc`: binding `"Chris Bizer"` for `?fn` and `dbpedia:Berlin` for `?b`. However, dereferencing the latter URI would be pointless: we do not need any information about `dbpedia:Berlin` to answer the query. Our optimisation thus proposes to avoid wasting lookups by not dereferencing URIs bound to non-join variables such as `?b`. □

By reducing the amount of sources and raw data that are accessed—and given that anyone in principle can say anything, anywhere—we may also reduce the number of answers that are returned. Taking the pre-

[12] http://jena.apache.org/documentation/query/
[13] http://code.google.com/p/ldspider/

vious example, for all we know, the document dereferenced through `dbpedia:Berlin` may contain people based near Berlin that Olaf knows, which would help to contribute other answers. However, we deem this to be unlikely in the general case, and note that it goes against the core LTBQE idea of using Linked Data principles to find query-relevant sources.

Aside from this optimisation to avoid dereferencing URIs bound to non-join variables, we note that URIs in certain positions of a triple pattern may not be worth dereferencing to look for matching information. For example, given the pattern "`?s foaf:knows ?o .`", we would not expect to find (m)any triples matching this pattern in the document dereferenced by `foaf:knows`. In the next section, we investigate precisely this matter for different triple positions, and thereafter propose a further variation on LiDaQ's source selection to prune remote lookups that are unlikely to contribute answers.

6. Empirical Study

LTBQE relies on the assumption that relevant data are dereferenceable, which may not always hold in practice. In this section, we analyse a large sample of the Web of Data to see what ratio of information is available in dereferenceable documents versus the total information available in the entire sample. This provides insights as to what percentage of raw data is available to LTBQE versus, *e.g.*, a materialised approach with a complete index over a large crawl of the Web of Data. We can also test how much additional raw information is made available by our extensions.

6.1. Empirical corpus

We take the dataset crawled for the Billion Triple Challenge 2011 (BTC'11) in mid-May 2011 as our corpus. The dataset consists of 7.4 million RDF/XML documents spanning 791 pay-level domains (data providers). URIs extracted from all RDF triples positions (excluding common non-RDF/XML extensions like `.pdf`, `.jpg`, `.html`, *etc.*) were considered for crawling. The resulting corpus contains 2.15 billion quadruples (1.97 billion unique triples) mentioning 538 million RDF terms, of which 52 million ($\sim$10%) are literals, 382 million ($\sim$71%) are blank nodes, and 103 million ($\sim$19%) are URIs. We denote the corpus as $\Gamma_\sim$. The core RDF data are serialised as N-Quads [13]: a syntax that extends N-Triples with a fourth element, used in this case to track which triple came from which source (following our notion of a Linked Dataset).

Alongside the RDF data, all relevant HTTP information, such as response codes, redirects, *etc.*, are made available. However, being an incomplete crawl, not all URIs mentioned in the data were looked up. As such, we only have knowledge of redir and deref functions for 18.65 million URIs; all of these URIs are HTTP and do not have non-RDF file-extensions. We denote these URIs by $U_\sim$. Of the 18.65 million, 8.37 million ($\sim$45%) dereferenced to RDF; we denote these by $D_\sim$.

Again, this corpus is only a *sample* of the Web of Data: we can only analyse the HTTP lookups and the RDF data provided for the corpus. Indeed, a weakness of our analysis is that the BTC'11 dataset only considers dereferenceable RDF/XML documents and not other syntaxes like RDFa or Turtle. Thus, our estimate of what ratios of relevant data are dereferenceable should be considered as an upper bound since there are many documents on the Web of Data that we do not (or cannot [57]) know about.

6.2. Static Schema Data

For the purposes of this analysis, we extract a static set of schema data for the RDFS reasoning. As argued in [10], schema data on the Web is often noisy, where third-party publishers "redefine" popular terms outside of their namespace. Thus, we perform authoritative reasoning, which conservatively discards certain third-party schema axioms (cf. [10]). In effect, our schema data only includes triples of the following form:

$$\begin{array}{lr}
\text{PRP-SPO1}: & (s, \texttt{rdfs:subPropertyOf}, o) \in \mathsf{deref}(s)\\
\text{PRP-DOM}: & (s, \texttt{rdfs:domain}, o) \in \mathsf{deref}(s)\\
\text{PRP-RNG}: & (s, \texttt{rdfs:range}, o) \in \mathsf{deref}(s)\\
\text{CAX-SCO}: & (s, \texttt{rdfs:subClassOf}, o) \in \mathsf{deref}(s)
\end{array}$$

We call these *authoritative schema triples*. Table 2 gives a breakdown of the counts of triples of this form extracted from the dataset, and how many domains (PLDs) they were sourced from: a total of 397 thousand triples were extracted from schema data provided by 98 PLDs. We denote this dataset as $\Psi_\sim$.

6.3. Recall for Baseline

We first measure the average dereferenceability of information in our sample. Let $\mathsf{data}(u, G)$ give the triples mentioning a URI u in a graph G, and, for a dereferenceable URI d, let $\mathsf{ddata}(d)$ denote $\mathsf{data}(d, \mathsf{deref}(d))$: triples dereferenceable through d

Table 2
Breakdown of authoritative schema triples extracted from the corpus

Category	**Triples**	**PLDs**
`rdfs:subPropertyOf`	10,902	67
`rdfs:subClassOf`	334,084	82
`rdfs:domain`	26,207	79
`rdfs:range`	26,204	77
total	397,397	98

mentioning d in some triple position. We then define the *sample dereferencing recall* for d w.r.t. G as:

$$\mathsf{sdr}(d, G) := \frac{\mathsf{ddata}(d)}{\mathsf{data}(d, G)}$$

Letting $G_\sim := \mathsf{merge}(\Gamma_\sim)$ denote the merge of our corpus, we measure $\mathsf{sdr}(d, G_\sim)$, which gives the ratio of dereferenceable triples for d mentioning d vs. unique triples mentioning d across the corpus. For comparability, we do not dereference d live, but use the HTTP-level information of the crawl to emulate $\mathsf{deref}(.)$ at the time of the crawl. We denote by $\mathsf{ddata}_\sim$ the average of $\mathsf{ddata}(d)$ for all $d \in D_\sim$, and by $\mathsf{sdr}_\sim$ the average of $\mathsf{sdr}(d, G_\sim)$ for all $d \in D_\sim$.

We also measure analogues of $\mathsf{ddata}_\sim$ and $\mathsf{sdr}_\sim$ where d must appear in specific triple positions: for example, if LTBQE dereferences a URI in the predicate position of a triple pattern, we wish to know how often relevant triples—*i.e.*, triples with that URI as predicate—occur in the dereferenced document, how many, and what ratio compared with the whole corpus.

Table 3 presents the results, where for different triples positions we present:

$\lvert U_\sim \rvert$	:	number of URIs in that position,
$\lvert D_\sim \rvert$	:	number of which are dereferenceable,
$\frac{\lvert D_\sim \rvert}{\lvert U_\sim \rvert}$	:	ratio of dereferenceable URIs
$\mathsf{sdr}_\sim$	:	as above, with std. deviation (σ)
$\mathsf{ddata}_\sim$	:	as above, with std. deviation (σ)

The row TYPE-OBJECT only considers the object position of triples with the predicate `rdf:type`, and the row OBJECT only considers object positions where the predicate is not `rdf:type`.

A number of observations are directly relevant to LTBQE. Given a HTTP URI (without a common non-RDF/XML extension), we have a $\sim$45% success ratio to receive RDF/XML content regardless of the triple position; for subjects, the percentage increases to $\sim$85%, *etc.* If such a URI dereferences to RDF, we receive on average (at most) $\sim$51% of all triples in which it appears across the whole corpus. Given a triple pattern with a URI in the subject position, the dereferenceable ratio increases to $\sim$95%, such that LTBQE would work well for (possibly partly bound) query patterns with a URI in the subject position. For objects of non-type triples, the ratio drops to 44%. Further still, LTBQE would perform very poorly for triple patterns where it must rely on a URI in the predicate position or a class URI in an object position: the documents dereferenced from class and property terms rarely contain their respective extension, but instead often contain schema-level definitions.

Table 3 also features high standard-deviation values: these indicate that dereferenceability is often "all or nothing". In relative terms, predicate and type-object deviations were the highest. Although most such terms return little or no relevant information—*e.g.*, dereferencing the predicate in a triple pattern rarely yields triples where the dereferenced term appears as predicate—we observed a few predicates and values for `rdf:type` return a great many relevant triples in their dereferenced documents.[14]

6.4. *Recall for Extensions*

We now study how much additional data is made available for query answering by the three LTBQE extensions. Table 4 presents the average increase in raw triples made available to LTBQE by considering `rdfs:seeAlso` and `owl:sameAs` links, as well as knowledge materialised through `owl:sameAs` and RDFS reasoning. $D^+_\sim$ indicates the subset of URIs in $D_\sim$ that have some relation to the extension, respectively: the URI has `rdfs:seeAlso` link(s), has `owl:sameAs` link(s), or has non-empty RDFS inferences. Also, $\mathsf{ddata}^+_\sim$ indicates the analogous $\mathsf{ddata}_\sim$ measure after the extension has been applied; *i.e.*, after relevant links are followed and/or inferences applied.

6.4.1. *Benefit of* `rdfs:seeAlso` *extension*

The percentage of dereferenceable URIs in $D_\sim$ with at least one `rdfs:seeAlso` link in their dereferenced document was $\sim$2% (about 201 thousand URIs).

[14]Many such examples for both classes and properties come from the SUMO ontology: see, *e.g.*, `http://www.ontologyportal.org/SUMO.owl#subsumingRelation` for a large extension of ontology terms provided by the ontology itself.

Table 3
Dereferenceability results for different triple positions

Position	$\lvert U_\sim\rvert$	$\lvert D_\sim\rvert$	$\frac{\lvert D_\sim\rvert}{\lvert U_\sim\rvert}$	$\mathsf{sdr}_\sim$		$\mathsf{ddata}_\sim$	
				avg.	σ	*avg.*	σ
ANY	1.87×10^7	8.37×10^6	0.449	0.51	±0.5	17.26	±97.15
SUBJECT	9.55×10^6	8.09×10^6	0.847	0.95	±0.19	14.11	±35.46
PREDICATE	4.77×10^4	745	0.016	0.00007	±0.008	0.14	±56.68
OBJECT	9.73×10^6	4.50×10^6	0.216	0.44	±0.46	2.95	±60.64
TYPE-OBJECT	2.13×10^5	2.11×10^4	0.099	0.002	±0.05	0.07	±29.13

Table 4
Additional raw data made available through LTBQE extensions

Extension	$\lvert D^+_\sim\rvert$	$\frac{\lvert D^+_\sim\rvert}{\lvert D_\sim\rvert}$	$\frac{\mathsf{ddata}^+_\sim}{\mathsf{ddata}_\sim}$	
			avg.	σ
SEEALSO	2.01×10^5	0.02	1.006	±0.04
SAMEAS	1.35×10^6	0.16	2.5	±36.23
RDFS	6.79×10^6	0.84	1.8	±0.76

Where such links exist, following them increases the amount of unique triples (involving the original URI) by a factor of $1.006\times$ versus the unique triples in the dereferenced document alone. We conclude that the `rdfs:seeAlso` extension will only marginally affect the recall increase of LTBQE in the general case.

6.4.2. *Benefit of* `owl:sameAs` *extension*

We measured the percentage of dereferenceable URIs in $D_\sim$ which have at least one `owl:sameAs` links in their dereferenced document to be $\sim 16\%$ for our sample. Where such links exist, following them and applying the EQ-* entailment rules over the resulting information increases the amount of unique triples (involving the original URI) by a factor of $2.5\times$ vs. the unique (explicit) triples in the dereferenced document alone. The very high standard deviation of ± 36.23 shown in Table 4 is explained by the plot in Figure 4 (log/log), which shows the distribution of the ratio of increase by considering dereferenceable `owl:sameAs` links and inferences for individual entities: we again see that although the plurality of entities enjoy only a small increase in raw data (close to the x-axis), a few entities enjoy a very large increase (farther from the x-axis). In more detail, Figure 5 gives a breakdown for URIs from individual domains, showing the number of URIs with an information increase above the indicated threshold due to `owl:sameAs`. The graph shows that, *e.g.*, some URIs from `nytimes.com` and `freebase.com` had an information increase of over $4000\times$ (mostly due to DBpedia links); often the local descriptions were "stubs" with few triples.

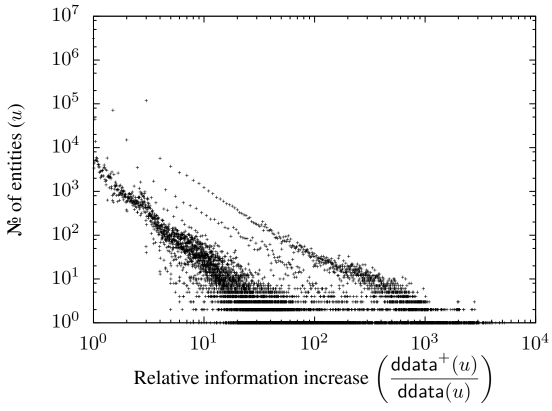


Fig. 4. Distribution of relative information increases by materialising `owl:sameAs` information (log/log)

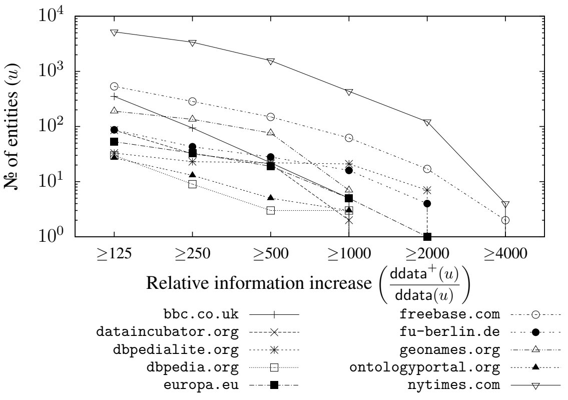


Fig. 5. Binning of relative information increases by materialising `owl:sameAs` information per domain (log/log)

We conclude that `owl:sameAs` links are generally not so common for dereferenceable URIs, but where available, following them and applying the entailment rules generates significantly more (occasionally orders of magnitude more) data for generating answers.

6.4.3. Benefit of RDFS extension

With respect to our authoritative static schema data $\Psi_{\sim}$, we measured the percentage of dereferenceable URIs in $D_{\sim}$ whose dereferenced documents give non-empty entailments as $\sim$81%. Where such entailments are non-empty, they increase the amount of unique triples (involving the original URI) by a factor of $1.8\times$ vs. the unique (explicit) triples in the dereferenced document. We conclude that such reasoning often increases the amount of raw data available for LTBQE query answering, and by a significant amount.

6.5. Discussion

Before looking at specific queries, in this section we find that, in the general case, LTBQE works best when a subject URI is provided in a query-pattern, works adequately when only (non-class) object URIs are provided, but works poorly when it must rely on property URIs bound to the predicate position or class URIs bound to the object position. Furthermore, we see that `rdfs:seeAlso` links are not so common (found in 2% of cases) and do not significantly extend the raw data made available to LTBQE for query-answering. Conversely, `owl:sameAs` links are a bit more common (found in 16% of cases) and can increase the available raw data significantly ($2.5\times$). Furthermore, RDFS reasoning often (81% of the time) increases the amount of available raw data by a significant amount ($1.8\times$).

As discussed previously, we can use these results to justify a variant of LTBQE that tries to minimise wasted remote lookups: aside from skipping URIs bound to non-join variables, and unless collecting schema data dynamically, this variant skips dereferencing predicate URIs bound in triple patterns, or URIs bound to the objects of triples patterns where the predicate is bound to `rdf:type`, since we are unlikely to find data matching those patterns in the respectively dereferenced document (*cf.* Table 3).

7. Query Benchmarks

We wish to evaluate LiDaQ in a realistic, uncontrolled environment: answering SPARQL queries directly over a diverse set of Web of Data sources. To guide this evaluation, we first survey existing Linked Data SPARQL benchmarks and look at how other systems evaluate their approaches (§ 7.1). We conclude that no benchmark offers a large and diverse range of benchmark SPARQL queries and thus propose *QWalk*: a novel benchmark methodology tailored for testing LTBQE-style query-answering approaches over the broader Web of Data (§ 7.2). Final evaluation setup and results are presented later in Section 8.

7.1. Existing Linked Data SPARQL Benchmarks

Since we aim to run our evaluation over Linked Data sources *in situ*, we ignore benchmarks designed to run over synthetic datasets such as *LUBM* [21], *BSBM* [9], or *SP^{2}Bench* [51, 52]. To the best of our knowledge, this leaves only two standard benchmarks:

FedBench [50] offers three data collections for testing Linked Data querying scenarios:

1. a Life Science Data Collection, which includes datasets like KEGG, ChEBI, DrugBank and DBPedia;
2. a synthetic dataset from the SP^{2}Bench framework [51, 52]; and
3. a general Linking Open Data Collection, which includes datasets like DBpedia, GeoNames, Jamendo, LinkedMDB, The New York Times and Semantic Web Dog Food.

A query set is defined for each. The first query set focuses on features of particular interest for federated query engines, such as the number of involved sources, (interim) query results size, and so forth. The second query set consists of the original SP^{2}Bench queries. The third set provides 11 Linked Data queries and is thus relevant to us.

DBSPB [43] (the DBpedia SPARQL Benchmark) contains SPARQL queries distilled from real-world DBpedia logs, consisting of 31.5 million queries issued by various users and agents over a four-month time-frame in 2010. The raw set of queries is reduced to a total of 35 thousand queries after less frequently-occuring query shapes were removed. These 35 thousand queries are clustered to generate 25 templates that characterise the larger set. These templates can be "instantiated" to create new queries from DBpedia data. The core of the templates consist of Basic Graph Pattern queries with 1–5 triple patterns, but may also include various combinations of SPARQL query features (*e.g.*, `OPTIONAL–FILTER–DISTINCT`, `UNION–FILTER`, and so forth).

A variety of Linked Data querying papers have also defined once-off evaluation frameworks. The methods used by a selection of papers are summarised in Table 5. We see that two approaches (FedX [53] and SPLENDID [20]) evaluate their methods using Fedbench, but do not run their queries live. Most papers simulate HTTP lookups or replicate SPARQL endpoints locally. We see that only two papers feature evaluation of queries that are run live over HTTP, both by Hartig [26,27], one involving six hand-crafted queries over real-world sources [27], the other proposing a "FOAF Letter" application to keep track of social connections where five query templates are instantiated for 23 people, giving a total of 115 queries.

In summary, there is much diversity in how Linked Data query proposals have been evaluated. Few benchmarks have been proposed for real-world Linked Data; perhaps the most agreed upon is FedBench. Most evaluations involve either a handful of custom queries designed to run over a small number of sources (FedBench), or a large number of (semi-)automatically generated queries tied to a specific domain (*e.g.*, DBPSB) or vocabulary (*e.g.*, FOAF Letter). Few engines run their queries live over remote resources, but instead replicate raw content or endpoints within a controlled environment. As such, no live Linked Data query engine has been evaluated in an uncontrolled, real-world setting for a large set of diverse queries: the closest such evaluation is probably FOAF Letter [26], but which only created queries over FOAF profiles.

7.2. *QWalk: Random Walk Query Generation*

Given the shortcomings of existing benchmarks, we propose *QWalk*: a new benchmark framework that automatically builds a large set of queries that are answerable (*e.g.,* by LTBQE) over a diverse set of real-world sources. The core idea is to take a large crawl of the Web of Data (in this case, the BTC'11 dataset) and to conduct random walks of different shapes and lengths through the corpus to generate Basic Graph Patterns. The walk is guided to ensure that it crosses documents through dereferenceable links: generated queries should thus be answerable by LTBQE.

7.2.1. *Query shapes*

To inform the types of queries we generate, we take observations from the work of Gallego *et al.* [18], who analyse the SPARQL queries logs of the DBPedia and Semantic Web Dog Food (SWDF) servers. They found that most queries contain a single triple pattern (66.41% in DBPedia, 97.25% in SWDF). The maximum number of patterns found was 15, but such complex queries occurred only rarely. The most common forms of joins involved subject–subject (59–61%), subject–object (32–36%) and object–object (4–5%); few joins involving predicate variables were found in general. As such, most queries with multiple patterns are star-shaped, with a few path shaped queries. Star-shaped joins typically had a low "fan-out", where 27% of the DBpedia queries had a fan-out of three, and 3.7% had a fan-out of two; the bulk of the remaining queries were single-pattern with a trivial fan-out of one, but went up to a maximum of nine. The lengths of paths in the query were mostly one (98%) or two (1.8%); very few longer paths were found.

Along similar lines, for our benchmark we generate queries of elemental graph shapes as depicted in Figure 6, *viz.,* edge, star and path queries. We now describe these query types in more detail.

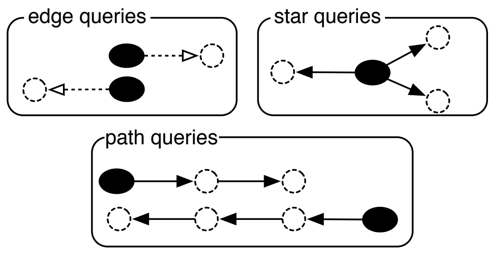


Fig. 6. Visualisation of example query shapes (edge-s, edge-o, s-path-2, o-path-3, star-2-1); dotted nodes represent variables; solid nodes represent URIs

Edge queries (edge-<s|o|so>) fetch all triples (edges) for an entity. We generate three sub-types of edge queries, asking for triples where a URI appears as the subject (edge-s); as the object (edge-o); as the subject *and* object (edge-so). These types of queries are very common in Linked Data browsers or for dynamically serving dereferenceable Linked Data content. An example query for edge-so would be:

```
SELECT DISTINCT ?p1 ?o ?s ?p2 WHERE {
  <d> ?p1 ?o . ?s ?p2 <d> . }
```

Star queries (star-<s3|o3|s2-o1|s1-o2>) consist of three acyclic triple patterns that share exactly one URI (called the centre node). These queries are similar to edge queries but have only constant predi-

Table 5
Summary of evaluation setups in the Linked Data querying literature

Reference	Queries			Measures			Live	Evaluation Setup
	count	*type*	*published*	*time*	*results*	*sources*		
LTBQE1 a) [29]	4	Custom	X	✓	✓	✓	✓	Single run
LTBQE1 b) [29]	12	Custom	X	✓	X	X	X	BSBM query mixes, RAP Pubby setup
LTBQE2 [31]	200	Custom	✓	✓	X	✓	X	BSBM query mixes, RAP Pubby setup
LTBQE3 [26]	115	Custom	✓	✓	✓	✓	✓	3 runs per query
LTBQE4 [27]	3	Custom	✓	✓	✓	✓	✓	6 runs per query (1st run warmup)
LT10 [39]	8	Custom	✓	✓	✓	✓	X	Controlled with 2 second delay proxy
SIHJoin [40]	10	Custom	✓	✓	X	X	X	CumulusRDF Linked Data proxy
FedX [53]	11	FedBench	✓	✓	X	X	X	Local copies of SPARQL endpoints
LH10 [42]	390	Custom	X	✓	✓	✓	X	Simulated HTTP lookups
SPLENDID [20]	14	FedBench	✓	✓	X	✓	X	Local replication of SPARQL endpoints
SPARQL-DQP [3]	7	Custom	✓	✓	X	X	X	Amazon EC2 instance
QTree [59]	300	Auto-gen.	X	✓	✓	✓	X	Simulated HTTP lookups

cates, asking for specific attributes of a given entity. Each query has 4 constants and 3 variables. We generate four sub-types of star queries, differing in the number of triple patterns in which the centre node appears at the subject (s) or object (o). An example for star-s2-o1 would be:

```
SELECT DISTINCT ?o1 ?o2 ?s1 WHERE {
  <d> foaf:knows ?o1 ; foaf:name ?o2 .
  ?s1 dc:creator <d> . }
```

Path queries (<s|o>-path-<2|3>) consist of 2 or 3 triple patterns with constant predicates that form a path such that precisely two triple patterns share a given join variable. Precisely two patterns contain one join variable, and the remaining patterns contain two join variables. Precisely one triple pattern has a URI at either the subject or object position. We generate four different sub-types: path shaped queries of length 2 and 3 in which either the subject or object term of one of the triple patterns is a constant. An example for s-path-2 is:

```
SELECT DISTINCT ?o1 ?o2 WHERE {
  dblpP:HartigBF09 foaf:maker ?o1 .
  ?o1 foaf:name ?o2 . }
```

Query generation We generate queries from the aforementioned BTC'11 dataset. In total, we generate 100 SELECT DISTINCT queries for *each* of the above 11 query shapes using random walks in our corpus. To help ensure that queries return non-empty results (in case there are no HTTP connection errors or time outs) we consider dereferenceable information and generate queries as follows:

1. We randomly pick a pay-level-domain available in the set of confirmed dereferenceable URIs $D_{\sim}$.
2. We then randomly select a URI from $D_{\sim}$ for that pay-level-domain.
3. We generate appropriate triple patterns from the dereferenceable document of the selected URI based on the query shape being generated.
   – If path-shaped queries are being generated, the URI for the next triple pattern is selected from the dereferenceable URIs connected to the previous URI, as per a random walk.
4. One variable is randomly chosen as distinguished (returned in the SELECT clause) and other variables are made distinguished with a probability of 0.5.

By randomly selecting a pay-level-domain first (as opposed to randomly selecting a URI directly), we achieve a greater spread of URIs across different datasets. The result of the QWalk process is a large set of diverse queries with different elemental shapes that—according to the sampled data—should be answerable through LTBQE methods over real-world data in a realistic scenario (accessing remote sources).

8. Query Benchmark Results & Discussion

When running queries directly over remote sources, various challenges come to the fore, including slow HTTP lookups, unpredictable remote server behaviour, high fan-out of links to traverse, the need for polite access (in terms of delays between lookups and respecting `robots.txt` policies), and so forth. As discussed, most works have evaluated LTBQE-style approaches in controlled environments using proxies or only for a small number of queries or sources in uncontrolled environments. Conversely, we now wish to investigate the feasibility of LTBQE—and our proposed reasoning extensions—for a broad range of diverse queries in uncontrolled environments so as to characterise how these methods cope with real-world challenges.

Based on the discussion of the previous section, we select three complementary benchmarks to run with LiDaQ, where we execute all queries directly over the Linked Data Web:

1. DBpedia SPARQL Benchmark (DBPSB), which generates many queries designed to run over one domain based on real-world query logs (we encountered difficulties running DBPSB with LTBQE as summarised later);
2. FedBench Linked Data Queries, which offers some manually crafted queries designed to run over a small selection of different domains;
3. QWalk, which offers a large selection of synthetic queries that can be run directly over a diverse set of sources.

We first discuss the experimental setup and measures (§ 8.1). Next we introduce the configurations of LiDaQ that we evaluate (§ 8.2). We then briefly summarise the problems we encountered when running DBPSB queries (§ 8.3). Finally, we present detailed results for FedBench (§ 8.4) and QWalk (§ 8.5).

8.1. *Experimental Setup and Measures*

All evaluation is run on one server: 4 GB RAM, Debian OS, 2.2GHz single core. To ensure polite behaviour, we enforce a per-domain (specifically per-PLD) minimum delay of 500 ms between two sequential HTTP lookups on one domain. Furthermore, we use a (generous) query timeout of 2 hours.

When running queries live over HTTP, we often encounter some "non-deterministic" behaviour: a source may be readily accessible during some query runs, but may be unresponsive or not return at all in others. Different HTTP-level issues can occur at different times for the same source. During initial experiments, we thus encountered that result size and execution time can differ for the same query and setup between several benchmark runs. This "inconsistent" query behaviour is explained by the fact that we encountered various HTTP-level issues between different executions for the same query and setup. We thus define the straightforward notion of a *benchmark stable query* to help with comparability across different setups. We assume that a query has a core set of relevant documents, which are accessed in all configurations (introduced in the following section). A *benchmark stable query* is a query for which the response codes for each of the core URIs is the same across all setups runs.

For each query in each evaluation framework, we record the number of unique **results** returned and the total **time** taken for the query to execute and terminate. We also record the **first** result latency, where a query may begin to return results quickly but may take a long time to terminate due to a few slow accesses for the final results. We record the number of **HTTP** lookups performed; since individual lookups are slow and we additionally implement politeness policies, this will be a major factor for performance issues. We also look at the raw **data** processed and number of triples **inferred**, both expressed as number of unique triples.

In initial experiments, we found that raw counts of unique query results sometimes exhibit outliers due to redundancy caused by joins. We illustrate the problem with an example QWalk query:

Example 11. Take the following QWalk query:

```
SELECT DISTINCT ?s0 ?o0 ?s1 WHERE {
  ebiz: owl:imports ?o0 .
  ?s0 rdfs:seeAlso ebiz: .
  ?s1 rdfs:isDefinedBy ebiz: . }
```

Without reasoning, upon dereferencing the `ebiz:` URI, we found 1 binding for `?o0`, 2 bindings for `?s0` and 199 bindings for `?s1`, yielding a total of $1 \times 2 \times 199 = 398$ results. However, with RDFS reasoning enabled, the schema document for RDFS (the so-called "`rdfs.rdfs`" document) authoritatively defines `rdfs:isDefinedBy` to be a sub-property of `rdfs:seeAlso`. Thus, the 199 bindings for `?s1` are added to `?s0`, yielding 201 bindings, and a total of $1\times201\times199 = 39,999$ results, giving a two orders of magnitude increase. □

The query results in this example contain many repetitions of terms: each term bound to `?s0` or `?s1` would appear about 200 times with RDFS reasoning enabled. Hence we add another measure to help characterise the "redundancy-free content" of the results: we add the number of unique result **terms** found for each distinguished variable in the results. In the above example, this would give $1 + 2 + 199 = 202$ terms without reasoning, and $1+201+199 = 401$ terms with reasoning.

8.2. LiDaQ Configurations Evaluated

For LiDaQ, given the various extensions, various ways of collecting RDFS data, and the option to turn off/on our reduction of sources, we have thirty-two combinations of possible setups, where we choose the following ten configurations to evaluate:

Core (CORE)**:** we dereference all URIs appearing in the query and during the query execution, independent from their triple position or role in joins. No extensions are included. This setup serves primarily as reference to the basic LTBQE profile.

Reduced Core (CORE^-)**:** Our reduced configuration uses our optimised source selection to decide which URIs are query relevant and should be dereferenced. We do not dereference URIs bound to non-join variables, or (unless dynamically retrieving schemata) URIs bound to predicates or values for `rdf:type`. In theory, we expect the smallest amount of results and also the fastest query time and the following extensions are built upon CORE^-, not CORE.

With `rdfs:seeAlso` links (SEEALSO)**:** this configuration extends the CORE^- setup by following `rdfs:seeAlso` links. Based on our empirical analysis, we expect that only a small number of queries will be affected given that $\sim 2\%$ of URIs have dereferenceable `rdfs:seeAlso` links and that following such links finds little relevant data.

With `owl:sameAs` links and inference (SAMEAS)**:** this benchmark setup extends the CORE^- setup by considering `owl:sameAs` links and inference. We expect an increase in returned results and the number of lookups. Based on our empirical study, this should affect a moderate number of queries given that such links are available for $\sim 16\%$ of resources, where, in such cases, inference makes on average $2.5\times$ more raw data available.

With RDFS inference $\left(\text{RDFS}_{[s|d|e]}\right)$**:** this benchmark setup extends the CORE^- setup by performing inferences for the RDFS ruleset relying on retrieved schema information. Based on static schema data, our empirical analysis suggested that RDFS reasoning affects $\sim 84\%$ of resources for which it makes about $1.8\times$ more raw data available. However, we investigate three sub-configurations based on the methods described in Section 5.1.3:

- **Static** (RDFS_s)**:** uses the *static schema* extracted earlier from the BTC'11 dataset;
- **Direct** (RDFS_d)**:** collects *direct schemata* by dynamically dereferencing predicates and values for `rdf:type`;
- **Extended** (RDFS_e)**:** collects *extended schemata* by dynamically following recursive links from the direct schemata.

Combined $\left(\text{COMB}_{[s|d|e]}\right)$**:** this benchmark setup combines all extensions for the previously mentioned configurations of schema collection. With this configuration, we expect the highest number of results, query time and processed and inferred statements.

For reference, Table 6 summarises these test configurations and which features are enabled or disabled.

Label	reduced sources	`rdfs:seeAlso`	`owl:sameAs`	static RDFS	direct RDFS	extended RDFS
CORE						
CORE^-	✓					
SEEALSO	✓	✓				
SAMEAS	✓		✓			
RDFS_s	✓			✓		
RDFS_d	✓				✓	
RDFS_e	✓					✓
COMB_s	✓	✓	✓	✓		
COMB_d	✓	✓	✓		✓	
COMB_e	✓	✓	✓			✓

Table 6
Overview of the ten LiDaQ benchmark configurations

8.3. DBPSB Result Summary

We first briefly summarise our experiences in trying to use DBpedia SPARQL benchmark (DBPSB) to test LTBQE and our extensions. DBPSB involves 25 query templates generated from real-world DBpedia query logs, which are not designed specifically to be run with LTBQE. Our experiments had broadly negative results. Full details of the experiments are available in [56, § 4.5.2.2]. To summarise, of the 25 query templates, we identified that 16 could not be run by

LTBQE, due to either containing an `OPTIONAL` clause[15] or only containing property/class URIs in the query (whose extension cannot be dereferenced in DBpedia).

Next we tried to generate 25 sample queries for each of the remaining 9 DBPSB templates. We ran the template queries provided for the benchmark against the public DBpedia SPARQL endpoint[16] and generated up to 1,000 results. From these, we randomly selected 25 results to generate the query instances. Of the 9 templates, we encountered problems instantiating another 3 where the endpoint either timed out or returned too few results. In running the queries generated by the remaining 6 templates[17], LTBQE could not return results for 3 templates due to (non-dereferenceable) RDF literals being generated in the queries.

The detailed discussion and results for the 3 remaining templates are available in [56, § 4.5.2.2]. To summarise, these three templates involved (1) listing the types of a given entity (DB1), (2) listing English comments and depictions or homepages for a given entity (DB13), and (3) a more complex query that asks for the French-language labels of French prefectures and German state capitals (DB17). The results for these three templates suggested that CORE$^-$ offers increased performance over CORE with no loss of results, but we encountered performance issues with SAMEAS where there are many outgoing `owl:sameAs` links from DBpedia that did not contribute answers to the DBpedia-specific queries of DBPSB. Also, although using static RDFS schema contributed many additional answers to the first query template, the dynamic import of schema proved expensive for DBpedia, where classes and properties dereference to separate documents.

The observation that so few query templates (based on real-world user logs) could be run by LTBQE is in itself a negative result and also makes it difficult to meaningfully interpret the results of our DBPSB experiments. Acknowledging this, we now rather focus on the results of the FedBench and QWalk benchmarks, which are designed with LTBQE in mind.

8.4. *FedBench results*

We now use the FedBench Linked Data Queries (§ 7.1) to measure the potential benefit of our proposed extensions and optimisations. These 11 cross-domain queries (denoted LD1–11) are designed to return a non-empty result set if executed over the Web with an LTBQE-style approach. The queries (along with results and discussion) can be found in Appendix A. Our first observation is that 4 out of the 11 manually crafted FedBench queries contain explicit `owl:sameAs` query patterns. This fact ties back with our initial motivation that including `owl:sameAs` information is important to answer queries across diverse sources. In the case of FedBench, these `owl:sameAs` relations are included explicitly; for our extension this would not be necessary (though we leave them in for comparability across LiDaQ configurations that include/exclude `owl:sameAs` inference).

Query Testing We had to make some amendments to the original 11 FedBench queries. First, since we count results, we add the `DISTINCT` solution modifier to all queries to eliminate duplicates. Second, during initial tests, we identified that 3 queries were not up-to-date with the DBpedia knowledge-base. These queries (LD8–10) used the predicate `skos:subject` where DBpedia now uses `dcterms:subject`. We directly replaced these predicates in the query. In one such query (LD9), we also added a missing `@en` language tag to the literal `"Luiz Felipe Scolari"` to correspond with DBpedia data. Our updated queries are online.[18]

Other queries returned no results. Two queries (LD6–7) required data from the `geonames.org` domain, which failed because the the `robots.txt` file on the subdomain[19] states that all agents are disallowed: we do not wish to contravene the Robots Exclusion Protocol. Query LD9 returns no results for any configuration (or any run) and query LD10 only sometimes returns results when `owl:sameAs` is considered: these are due to mismatches between the given queries and remote data that we could not easily fix without dramatically changing the original query (see Appendix A for details). Aside from LD7, which *only* involved the GeoNames domain, we run the other queries for timings, even if they do not return results.

Overview of Experiments For all benchmark configurations, we executed each FedBench query live over the Web once a week for four weeks. In Appendix A, for each individual query, we provide detailed results and discussion (selecting the best of the four runs).

[15]In SPARQL, `OPTIONAL` can be combined with a `!BOUND(.)` filter to emulate negation-as-failure, which cannot be run over an open Web of Data (unless a closed-dataset semantics is considered [28]).

[16]`http://dbpedia.org/sparql/`

[17]All template instances are available online: `http://code.google.com/p/lidaq/source/browse/queries/dbpsb.swj.25.tar.gz`

[18]`http://code.google.com/p/lidaq/source/browse/queries/fedbench.txt`

[19]See `http://sws.geonames.org/robots.txt`

We also include per-query comparison to the existing SQUIN library for LTBQE [29], which we generally find to be considerably faster, but which, to the best of our knowledge, does not include politeness policies; we thus exclude it for later larger-scale experiments.

Detailed Results Given the number of queries (10), configurations (11) and measures (6), we leave detailed discussion of results for individual FedBench queries to Appendix A.

Summarising, we observe that LTBQE works well for some simpler FedBench queries, but struggles in an uncontrolled environment for complex queries that require accessing a lot of sources. For example, even in the baseline CORE$^-$ configuration, LD11 required performing 1,125 lookups, and the most complex configuration—COMB$_e$—attempted 17,996 lookups. Relatedly, we generally observe that LTBQE extensions perform well for simple queries, but exacerbate performance issues for complex queries. In fact, although RDFS and owl:sameAs extensions work well on domains like data.semanticweb.org, configurations that involve following same-as or schema links struggle for data-providers such as DBpedia, which offer many such links (both internal and external). On a more positive note, CORE$^-$ often offers significant time savings over CORE with minimal effect on result sizes.

In addition, results sizes can become quite large (*e.g.*, LD11 returns 196,448 results in one configuration): the given SPARQL queries often contain numerous result variables in the SELECT clause (*e.g.*, 5 for LD11) and results can contain high redundancy. For example, DBpedia often contains a large number of labels for resources in different languages, where asking for the labels of result resources may multiply the raw result sizes by a factor of ten or more and where such behaviour has a cumulative effect.

In general, we would also expect that the results given by CORE$^-$ should be fewer or equal than for all other configurations, which are monotonic extensions; similarly, we would expect equal or more results in COMB$_x$ than RDFS$_x$, SAMEAS and SEEALSO (where x is one of the schema configurations). This expectation held true in practice for a number of the earlier queries (LD1–3), where for queries LD1 and LD3, the various extensions, including reasoning, found many more results. This monotonic increase also held true for certain other cases (*e.g.*, with the exception of COMB$_e$, LD4 shows this behaviour). However, it did not hold true in later queries: even selecting the best run from a span of four weeks, the unrepeatability of results played a major role in this evaluation. We thus now focus on characterising this issue.

Reliability Results Running complex queries live over networks of remote sources raises the question of reliability and repeatability. We now focus on how the results varied across the four runs to get a better idea of the repeatability of LTBQE/LiDaQ in a realistic setting. We summarise the average number of results and the corresponding standard deviation for each configuration and query across all four runs in Table 7.

LD11 in particular shows some unreliable behaviour across the four runs, where we estimated the absolute deviations to be between ∼28–140% of the mean, depending on the LiDaQ configuration: as aforementioned, this query required between 1,103–17,996 lookups. With this exception aside, across all other queries, the CORE, CORE$^-$ and SEEALSO configurations access the fewest sources and produce reliable results across the four runs. The results for the other variations—which include reasoning extensions—are less reliable in general. LD5 and LD10 show high deviations in the number of results returned for SAMEAS, LD5 shows high deviations for dynamic schema configurations, and LD4 shows high deviations for configurations involving RDFS reasoning (though not for combined configurations). In terms of absolute deviation as a percentage of the mean, we computed that the results for the other setups vary somewhere between ∼2–11% for most of the queries.

Conclusions When running the queries live over remote sources, we see complex and unpredictable behaviour across different configurations and across time: remote sources may give different responses at different times (*e.g.*, may give 50x errors during high server load), and the failure of an important source may break traversal at that point. We also see that reasoning extensions, particular those involving dynamic schema collection, often make the query behaviour more unreliable by trying to access more sources.

The FedBench queries predominantly request data from a few central sites: the first four queries (LD1–4) are based around the data.semanticweb.org data provider, and provide generally stable results. Other queries also rely on the hosts www4.wiwiss.fu-berlin.de and dbpedia.org and generally demonstrate less stable behaviour. Taking the former domain, for example, owl:sameAs extensions can cause erratic behaviour, potentially due to errors in how the relation is used for the DailyMed and LinkedCT datasets. For DBpedia, the schema descriptions of class and property terms

Table 7

Average result size and standard deviation across four query runs

Setup	**LD1**		**LD2**		**LD3**		**LD4**		**LD5**	
	avg.	σ	*avg.*	σ	*avg.*	σ	*avg.*	σ	*avg.*	σ
Core	333	±0	185	±0	191	±0	50	±0	21.5	±24.83
Core$^-$	333	±0	185	±0	190.75	±0.5	50	±0	24	±22.32
seeAlso	333	±0	185	±0	190.75	±0.5	50	±0	21.5	±24.83
sameAs	527.25	±3.95	185	±0	908.25	±35.53	146	±0	67.75	±135.5
RDFS$_s$	380	±0	185	±0	246	±0	50	±0	17.5	±21.24
RDFS$_d$	380	±0	185	±0	246	±0	50	±0	20.25	±23.41
RDFS$_e$	380	±0	185	±0	246	±0	37.5	±25	8	±9.38
Comb$_s$	674.5	±14.15	185	±0	1,385	±161.85	137	±92.57	0	±0
Comb$_d$	662.5	±14.71	185	±0	1,428	±122.36	151.25	±100.85	3.5	±7
Comb$_e$	674.5	±14.15	185	±0	1,428	±122.36	88.5	±59.29	—	—
Setup	**LD6**		**LD8**		**LD9**		**LD10**		**LD11**	
	avg.	σ	*avg.*	σ	*avg.*	σ	*avg.*	σ	*avg.*	σ
Core	0	±0	9.5	±10.97	0	±0	0	±0	21,801.75	±6,255.72
Core$^-$	0	±0	9.5	±10.97	0	±0	0	±0	14,322	±12,237
seeAlso	0	±0	19	±0	0	±0	0	±0	19,096	±9,373.88
sameAs	0	±0	10,535.5	±12,165.35	0	±0	2,512.5	±2,973.81	8,466	±10,940.75
RDFS$_s$	0	±0	9.5	±10.97	0	±0	0	±0	1,745.5	±3,491
RDFS$_d$	0	±0	1	±2	0	±0	0	±0	2,757.25	±3,508.19
RDFS$_e$	0	±0	3	±6	0	±0	—	—	—	—
Comb$_s$	0	±0	14,096.75	±16,295.82	0	±0	812.25	±1,624.5	71,438	±21,634.8
Comb$_d$	0	±0	—	—	0	±0	0	±0	177,596.5	±61,929.13
Comb$_e$	0	±0	—	—	—	—	—	—	50,660	±71,289.96

are hosted in individual documents and often interlink with related sources like YAGO and CYC, leading to unstable behaviour for dynamically retrieving schemata where many documents need to be recursively retrieved. Given that the queries are restricted to a few domains, the required politeness delays are a major factor for performance.

In summary, from the 11 original FedBench queries, which were designed to be run using LTBQE-style approaches, 4 queries show promising results, 3 return no results (2 involving access disallowed by `robots.txt`), and the remaining 4 queries show unpredictable behaviour across different runs and configurations. Some of the more complex queries involve accessing thousands and tens of thousands of sources at runtime. By requesting even more sources, our proposed reasoning extensions can aggravate reliability issues. This calls into question the practicality of the LTBQE approach (and our reasoning extensions) in uncontrolled environments for complex queries that span multiple sites and require many sources to answer.

8.5. QWalk results

Having briefly looked at the DBPSB queries, which target only the DBpedia domain; and having looked in more detail at the FedBench queries, which target a handful of domains; we now look at the results of the QWalk benchmark (§ 7.2), which builds a large set of queries answerable over a wide range of real-world sources. Using random walk techniques over the BTC'11 corpus, we created 100 queries for each of the 11 elemental shapes of the QWalk benchmark, giving 1,100 initial queries. We then ran these live over remote sources using various configurations of LiDaQ.

Query testing We first wish to filter out queries that did not return any answers or that did not show benchmark stable behaviour. To begin, for the edge query classes, we look at how many queries return empty results, how many return stable non-empty results suitable for comparison, and how many return unstable non-empty results (see § 8.1). Our notion of stability is measured across all ten configurations of LiDaQ, including the dynamic schema import extensions. We also looked at the breakdown of stable/unstable/empty results turning off the dynamic schema import (*i.e.*, turning off RDFS$_d$, RDFS$_e$, Comb$_d$, Comb$_e$). The results are shown in Table 8. Though the stability of edge-o and edge-so queries are not significantly affected, the number of stable queries for edge-s queries more than halves. Thus, due to these problems with instability and also long runtimes, and given the number of queries in the benchmark, we do not run the dynamic schema configurations for QWalk queries.

Table 8
Stable edge queries with and without dynamic schema extensions

Template	Total	Stable	
		wo/dyn.	*w/dyn.*
edge-s	100	60	27
edge-o	100	57	53
edge-so	100	59	54

Considering CORE$^-$, CORE, SEEALSO, SAMEAS, RDFS$_s$ and COMB$_s$ configurations for each query shape, Table 9 provides a breakdown of the total number of queries that return some results and exhibit stable or unstable behaviour, as well as the number of queries with no results. Typewritten numbers correspond to categories for HTTP server response codes encountered for queries with no results; the column "*mix*" indicates at least two different response codes and the column "*data*" indicates that the missing results are not related to URI errors and we assume that the remote data changed. We select only non-empty and stable queries for our comparison.

Table 9
Summary of stable/unstable/empty queries for QWalk benchmark

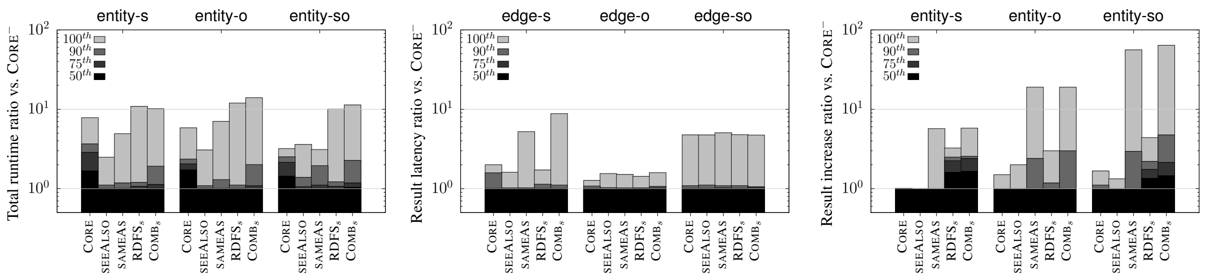
Class	Non-Empty		Empty					
	s.	*uns.*	*all*	*4XX*	*5XX*	*6XX*	*mix*	*data*
edge-o	57	7	36	18	2	11	0	5
edge-s	60	5	35	18	1	11	0	5
edge-so	59	9	32	17	2	8	1	4
o-path-2	62	4	34	16	5	8	1	4
o-path-3	35	25	40	19	3	18	0	0
s-path-2	66	2	32	17	2	11	0	2
s-path-3	51	7	42	18	1	20	0	3
star-0-3	67	6	27	14	0	10	0	3
star-1-2	62	2	36	21	2	12	0	1
star-2-1	70	5	25	11	3	9	1	1
star-3-0	66	15	19	12	0	4	0	3

Detailed Results We now look at the average measures for results across all (non-empty stable) queries per query class: we begin with edge queries, then progress to star queries and eventually to path queries. Detailed results for each of our measures can be found for reference in Table 21 of Appendix B. Herein, we plot the total runtime, result latency and result sizes in bar plots, where we measure the ratio of the analogous figure for CORE$^-$ (which always returns the fewest results and should be the fastest). Absolute measures can be found in Table 21. We generally found a lot of variance and outliers in our results; hence we plot the 50^{th}, 75^{th}, 90^{th} and 100^{th} percentiles to help show how the measures varied in the median case (given by the 50^{th} percentile) and for extremes/outliers (given by higher percentiles).

edge-*: GET ALL EDGES FOR A GIVEN RESOURCE
Edge queries have the most simple query shape and are used in a wide range of applications to gather all available information about a certain entity, *e.g.*, for the user interfaces of entity search engines. They are also often (but not always) used as a simple mechanism to support SPARQL `DESCRIBE` queries. Figure 7a presents the increase in time over CORE$^-$ for all other configurations across the three classes of edge queries, broken down by percentiles, with the x-axis presented in log-scale, where the 10^0 line indicates no change from CORE$^-$. Figure 7b analogously presents the increase in result latency (time taken to generate the first result) and Figure 7c presents the increase in query results returned versus CORE$^-$.

We can see from the 50^{th} percentile in Figure 7a that the CORE configuration—which dereferences predicates, values for `rdf:type` and URIs bound to non-join variables—often requires significantly more time to process queries than CORE$^-$ across all three edge query classes, with the most severe case (on the 100^{th} percentile) taking almost eight times longer for edge-s. Conversely, Figure 7c shows that CORE almost never returns additional results beyond CORE$^-$.

We also see that SEEALSO rarely affects performance, but very rarely finds additional answers (only for the 100^{th} percentile are result increases visible). From the flat 75^{th} percentiles in Figure 7a, we can see that in the majority of cases, other extensions did not affect performance significantly, and the time taken to start returning results was only affected significantly in outlier cases. However, for total runtimes, the 90^{th} and 100^{th} percentiles with reasoning show occasional increases by a factor of ten, but these are offset by increased result sizes, where modest increases are already visible on the 50^{th} percentile for RDFS$_s$ and COMB$_s$ in the edge-s and edge-so queries: all RDFS rules offer additional data for edge-s* queries, whereas only sub-property reasoning offers additional results for edge-o in the general case. The 75^{th}–100^{th} percentiles also show occasional but very large increases for results in the SAMEAS configuration.



(a) Total runtimes (b) Result latency (c) Result sizes

Fig. 7. Percentiles for ratio of increase in total runtimes, result latency and result sizes vs. CORE$^-$ for edge-query classes (log)

star-*: RETRIEVE VALUES FOR SPECIFIC PREDICATES ABOUT A GIVEN RESOURCE

Star-shaped queries retrieve selected attributes of an entity. The results for star-shaped queries are presented in Figure 8 following the same format as before.

Some similar conclusions can again be drawn as for edge queries. Again, the query time can often be reduced without a significant effect on query results by opting for CORE$^-$ over CORE; in this case, since query predicates are set, the savings are made by not dereferencing values for `rdf:type` or URIs bound to non-join variables. The result latency remains relatively stable except for in the 100^{th} percentile outlier. The notable total-runtime outlier for the query class star-1-2 in CORE (on the 100^{th} percentile) is due to one query which took around 1 hour to terminate because of the download and processing of a very large document from the `ecowlim.tfri.gov.tw` provider (such cases show the importance of also considering the result latency: this source contributed no results and was not accessed by configurations built on top of CORE).

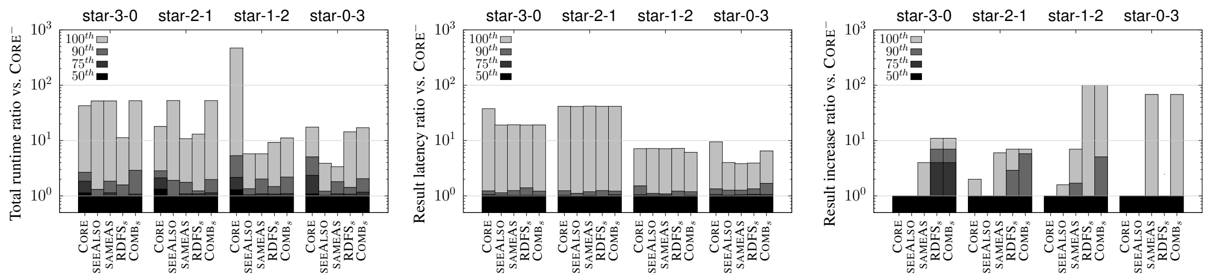
Again, we see that SEEALSO had minimal effect on results returned, but did increase the time significantly for some of the query classes. RDFS and `owl:sameAs` reasoning had an occasional but significant effect on results size: however, we highlight that the baseline gave, on average, very few results for star-3-0 and star-0-3 (*cf.* Table 21), where a small absolute increase could account for a very large relative increase, as per the outliers on the 100^{th} percentile. Also, the large RDFS-related results outlier for the class star-1-2 is due to the query mentioned in Example 11.

-path-: RETRIEVE TERMS THAT ARE TWO OR THREE HOPS AWAY FROM A CENTRAL RESOURCE THROUGH A PATH OF GIVEN PREDICATES

Path-shaped queries allow for exploring recursive relations in the graph, or to get information about neighbouring nodes. When compared with edge and star queries, we would expect path queries to generally be more expensive for LTBQE to process since they explicitly require traversing a number of sources.

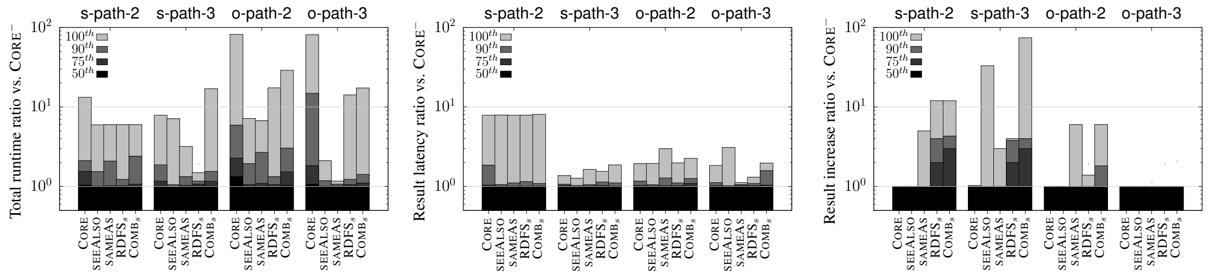
For path queries, we again show the analogous increases in runtimes for the five configurations in Figure 9. Again, we see the savings in time for selecting CORE$^-$ over CORE, particularly for the o-path-* classes of queries. We also see that result latencies are generally not affected by extensions, other than in the case of outliers. Across all extensions, the performance hit for o-path-* queries are not met with gains in results; in fact, the o-path-3 queries saw no significant gains for any extension, even for the 100^{th} percentile. We see the first meaningful gain for SEEALSO in the s-path-3 class, but only for a single outlier query. In this case, SAMEAS offers only minimal increases in some outlier cases. However, the RDFS$_s$ extension finds additional results for s-path-* queries, which are notable already on the 75^{th} percentile; this extension performs particularly well for s-path-3 where large gains in results do not cost comparable increases in total runtimes. The COMB$_s$ configuration again offers the most results, but—with the exception of CORE—at the cost of the highest runtimes.

Conclusion Across the hundreds of queries run for the 11 query classes, we consistently find that compared with CORE, the CORE$^-$ configuration saves significantly on total query runtimes with only minimal impact on result sizes. With the exception of one query, we find that SEEALSO finds barely any additional results, but can sometimes cause a significant increase in runtime. Reasoning extensions also increase total runtimes, but regularly contribute additional answers: SAMEAS offers infrequent but very high increases in result sizes, where by comparison, RDFS$_s$ offers more frequent but more modest increases in results. These



(a) Total runtimes (b) Result latency (c) Result sizes

Fig. 8. Percentiles for ratio of increase in total runtimes, result latency and result sizes vs. CORE$^-$ for star-query classes (log)



(a) Total runtimes (b) Result latency (c) Result sizes

Fig. 9. Percentiles for ratio of increase in total runtimes, result latency and result sizes vs. CORE$^-$ for path-query classes (log)

observations on result increases for the three extensions correspond well with the results of our analysis for the BTC'11 data in Section 6. Throughout, with the frequent exception of CORE, the combined approach was indeed the slowest, but always offered the most results. Although the reasoning extensions often significantly increased the time taken for queries to terminate, the latency for retrieving the first results was not significantly increased except in extreme cases.

To help summarise these results, Table 10 presents the average throughput (results per second) achieved across all queries per query class.[20] We see that CORE has uniformly the worst throughput of results across all query classes. We also see that CORE$^-$ generally performs slightly above average, but performs best for edge-o. With the sole exception of the s-path-3 class, SEEALSO performs slightly worse than CORE$^-$. In terms of the reasoning extensions, of the 11 query classes, the highest throughput for 9 are split between the SAMEAS (4), RDFS$_s$ (4) and COMB$_s$ (1) configurations, where, for each configuration, the throughput of COMB$_s$ frequently sits between SAMEAS and RDFS$_s$. However, aside from CORE, these latter configurations also often perform the worst: they add significant overhead to the query execution, but may often find significantly many additional results: they offer high-risk but high-gain.

Table 10

Results per second for all query classes with configurations shaded from best (lightest) to worst (darkest) throughput

	CORE	CORE$^-$	SEEALSO	SAMEAS	RDFS$_s$	COMB$_s$
edge-s	1	1.68	1.67	2.15	1.29	1.53
edge-o	3.97	6.48	6.16	5.7	5.37	4.38
edge-so	2.02	2.82	2.66	3.71	3.73	4.82
star-3-0	0.11	0.16	0.15	0.15	0.24	0.2
star-2-1	0.58	1.12	1	1.04	2.14	1.75
star-1-2	0.17	1.6	1.35	1.6	70.97	58.85
star-0-3	0.18	0.35	0.33	0.94	0.24	0.68
s-path-2	0.44	0.72	0.68	0.7	0.83	0.78
s-path-3	1.76	2.45	2.56	2.46	2.43	2.1
o-path-2	1.38	8.39	7.76	10.55	6.36	6.89
o-path-3	0.95	5.7	5.84	6.08	5.04	4.68

[20]Given that there is a lot of variance in the raw figures, we acknowledge that average figures are a coarse way to present the results, but they do help to summarise overall trends.

9. Conclusion

In theory, link-traversal query approaches for Linked Data have the benefit of up-to-date results and decentralised execution. However, a thorough evaluation of such methods in realistic uncontrolled environments—for a diverse Web of Data—had not yet been conducted. We have focused on evaluating LTBQE approaches in this manner and have investigated the possibility of combining lightweight reasoning methods with LTBQE to gather and integrate data from diverse sources during query-answering.

We have characterised what percentage of data is missed by only considering dereferenceable information, we have looked at what percentage of raw data is made available to LTBQE through various extensions, and we have tested LTBQE and various extensions in uncontrolled environments for three complimentary query benchmarks. Our results show that LTBQE works well for simple queries with a dereferenceable subject, but, in uncontrolled environments, struggles for more complex queries that involve accessing many remote sources at runtime. Furthermore, we showed that runtimes in uncontrolled environments are often a factor of politeness policies, since queries frequently access many documents from few domains.

In terms of the extensions, we have shown that the selection of sources can be successfully reduced by ignoring predicate URIs, object URIs for type-triples, and URIs bound to non-join positions. We have also shown that the `rdfs:seeAlso` extension offers little in terms of results, but occasionally introduces significant runtime costs. We also showed that `owl:sameAs` extensions can occasionally increase the number of results found by a great deal, but also comes at significant costs and introduces unstable behaviour when run live over domains such as DBpedia. Similarly, we showed that RDFS reasoning extensions increase results more frequently than `owl:sameAs` extensions (*e.g.*, in lower percentiles of the QWalk experiments), but exhibits more moderate increases than the latter extensions (*e.g.*, in the 100^{th} percentiles of QWalk experiments). Through the FedBench experiments, we also showed that the dynamic import of RDFS data at runtime works well for simple queries on certain domains (*e.g.*, `data.semanticweb.org`), but can introduce instability for domains such as DBpedia, where schemata are spread across multiple documents and link to other domains with similar decentralised schema.

Future Directions The combination of reasoning and LTBQE has shown the potential to find additional answers, and at a higher rate than without reasoning, but with the potential to make query-answering unstable. At the moment we focus on very lightweight reasoning, supporting an important subset of the semantics inherent in published Linked Data. Extending the inference rules to support a broader selection of OWL features—based on the observations of use by Glimm *et al.* [19]—would obviously help to find more answers. However, even for our lightweight reasoning, we already encounter practical problems. In particular, we showed that following `owl:sameAs` links caused problems for some queries that in baseline setups already involve many sources from the DBpedia domain, which offers a high density of `owl:sameAs` links from its local data. Furthermore, we noted that the dynamic import of RDFS data increased the complexity of remote access at runtime (esp. for DBpedia) and thus caused instability and inflated response times for more complex queries. Thus, we proposed to use static schema data where we assume that such data are infrequently updated. A better alternative—one that we did not investigate—is the use of lazy schema caching in combination with active refresh policies. We believe caching schema data would work well since a few (meta-)vocabularies (such as RDF, RDFS, OWL, FOAF, DC, DCTERMS, *etc.*) are used extremely frequently—something similar to a power-law driven by preferential attachment—and are generally quite static. Again, caching has obvious benefits for LTBQE in general (not just for schema), but herein we rather focus on query-at-a-time evaluation.

In general, due to various fundamental (*e.g.*, no support for `OPTIONAL`, *etc.*) and practical issues (reliance on dereferenceability, assumptions that query-patterns connect relevant sources through dereferenceable URIs, slow access to remote sources, varying stablity of remote hosts) LTBQE cannot be considered a complete *solution* for running complex SPARQL queries over Linked Data: SPARQL is simply too complex a query language to be supported in its entirety and in a practical fashion by LTBQE. As such, one may consider a different language for navigational queries, along the lines of proposals by Fionda *et al.* [17]. In general, a query language that would allow for declaratively specifying navigational aspects of query execution—*e.g.*, stick to the `data.semanticweb.org` domain, follow `foaf:knows` links, do not follow `foaf:homepage` links, *etc.*—would be interesting, and would allow users to better guide the query-engine than using a simple SPARQL query.

Taking an alternative view, although not a *solution* for SPARQL, LTBQE is an interesting *technique* for SPARQL and is complementary to other techniques for querying Linked Data, such as materialised or federated approaches. LTBQE offers the potential to get fresh answers when dynamic information is involved, or to get sensitive data when user-specific access-control is in place for some Linked Data source; this is not possible through centralised approaches. Furthermore, it does not rely on SPARQL interfaces like federated approaches; also, there are currently no mechanisms to *discover* endpoints in the same manner that LTBQE discovers sources. As such, the greatest potential for LTBQE is in combination with other querying techniques, for example to dynamically freshen-up results returned by a centralised SPARQL endpoint that replicates remote content. We have already begun to investigate this use of LTBQE—as a wrapper for the public LOD Cache and Sindice SPARQL endpoints–such that query patterns involving dynamic data are delegated to LTBQE rather than to replicated indexes, which are likely to be stale [60]. In such scenarios, LTBQE is required to deal with simple sub-queries, which we have shown to be feasible in this paper.

As the Web of Data continues to expand and diversify, and as it becomes more dynamic, new querying techniques will be required to keep up with its developments. Though various Web search engines have shown the power and potential of centralisation, even the preeminent Google machinery struggles to give up-to-date answers over dynamic sources. Linked Data presents new opportunities in this regard: URI names appearing in queries also correspond to addresses from which up-to-date data can be found. Although centralised approaches will always be relevant—a point of view which this paper partly confirms—exploring and combining complementary Linked Data querying techniques is an important area of research if we are to meet future challenges. In this paper, we have studied the realistic strengths and weaknesses of the LTBQE approach and various extensions. Next steps are to further explore how it can be combined with centralised query engines in an effective manner to freshen up answers over dynamic data, or to find answers from data-sources outside of the coverage of cached data.

Links Our source code and queries are available at `http://code.google.com/p/lidaq/wiki/Lidaq`.

Acknowledgements We would like to thank the reviewers for helping us to improve the paper and we would also like to thank Olaf Hartig for helpful discussion. The research presented herein has been supported in part by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289, by the Vienna Science and Technology Fund (WWTF) through project ICT12-015, and by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004.

References

[1] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: An adaptive query processing engine for SPARQL endpoints. In *ISWC*, pages 18–34, 2011.

[2] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Sem.*, 7(2):57–73, 2009.

[3] C. B. Aranda, M. Arenas, and Ó. Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *ESWC*, pages 1–15, 2011.

[4] C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In *ISWC*, pages 277–293, 2013.

[5] M. Arenas, C. Gutierrez, D. P. Miranker, J. Pérez, and J. Sequeda. Querying Semantic Data on the Web? *SIGMOD Record*, 41(4):6–17, 2012.

[6] T. Berners-Lee. Linked Data. Design issues, W3C, 2006.

[7] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):33–42, 2011.

[8] B. Bishop, A. Kiryakov, D. Ognyanov, I. Peikov, Z. Tashev, and R. Velkov. FactForge: A fast track to the Web of data. *Semantic Web*, 2(2):157–166, 2011.

[9] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[10] P. A. Bonatti, A. Hogan, A. Polleres, and L. Sauro. Robust and scalable Linked Data reasoning incorporating provenance and trust annotations. *J. Web Sem.*, 9(2):165–201, 2011.

[11] P. Bouquet, C. Ghidini, and L. Serafini. A formal model of queries on interlinked RDF graphs. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.

[12] G. Cheng and Y. Qu. Term dependence on the Semantic Web. In *ISWC*, pages 665–680, 2008.

[13] R. Cyganiak, A. Hogan, and A. Harth. N-Quads: Extending N-Triples with context. http://sw.deri.org/2008/07/n-quads/.

[14] R. Delbru, G. Tummarello, and A. Polleres. Context-dependent OWL reasoning in Sindice – experiences and lessons learnt. In *RR*, pages 46–60, 2011.

[15] L. Ding, J. Shinavier, Z. Shangguan, and D. L. McGuinness. SameAs networks and beyond: Analyzing deployment status and implications of owl:sameAs in Linked Data. In *ISWC*, pages 145–160, 2010.

[16] O. Erling and I. Mikhailov. RDF support in the Virtuoso DBMS. In *Networked Knowledge – Networked Media*. Springer, 2009.

[17] V. Fionda, C. Gutierrez, and G. Pirrò. Semantic navigation on

the Web of Data: specification of routes, Web fragments and actions. In *WWW*, pages 281–290, 2012.

[18] M. A. Gallego, J. D. Fernández, M. A. Martínez-Prieto, and P. De La Fuente. An empirical study of real-world SPARQL queries. In *USEWOD*, 2011.

[19] B. Glimm, A. Hogan, and M. Krötzsch. OWL: Yet to arrive on the Web of Data? In *LDOW*, 2012.

[20] O. Görlitz and S. Staab. SPLENDID: SPARQL endpoint federation exploiting voiD descriptions. In *COLD*, 2011.

[21] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2–3):158–182, 2005.

[22] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson. When owl:sameAs isn't the same: An analysis of identity in Linked Data. In *ISWC*, 2010.

[23] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C Recommendation, Mar. 2013.

[24] A. Harth and S. Speiser. On completeness classes for query evaluation on linked data. In *AAAI*, 2012.

[25] A. Harth, J. Umbrich, A. Hogan, and S. Decker. YARS2: A federated repository for querying graph structured data from the Web. In *ISWC*, 2007.

[26] O. Hartig. How caching improves efficiency and result completeness for querying Linked Data. *LDOW*, 2011.

[27] O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *ESWC*, 2011.

[28] O. Hartig. SPARQL for a Web of Linked Data: Semantics and computability. In *ESWC*, 2012.

[29] O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL queries over the Web of Linked Data. In *ISWC*, 2009.

[30] O. Hartig and J. C. Freytag. Foundations of traversal based query execution over Linked Data (extended version). *CoRR*, abs/1108.6328, 2011.

[31] O. Hartig and F. Huber. A main memory index structure to query Linked Data. *LDOW*, 2011.

[32] O. Hartig and A. Langegger. A database perspective on consuming Linked Data on the Web. *Datenbank-Spektrum*, 2010.

[33] P. Hayes. RDF semantics. W3C Recommendation, Feb. 2004.

[34] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011.

[35] A. Hogan, A. Harth, and A. Polleres. Scalable Authoritative Owl Reasoning for the Web. *Int. J. Semantic Web Inf. Syst.*, 5(2):49–90, 2009.

[36] A. Hogan, A. Zimmermann, J. Umbrich, A. Polleres, and S. Decker. Scalable and distributed methods for entity matching, consolidation and disambiguation over Linked Data corpora. *J. Web Sem.*, 10:76–110, 2012.

[37] K. Hose, R. Schenkel, M. Theobald, and G. Weikum. Database foundations for scalable RDF processing. In *Reasoning Web*, pages 202–249, 2011.

[38] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing Linked Data Dynamics. In *ESWC*, pages 213–227, 2013.

[39] G. Ladwig and T. Tran. Linked Data query processing strategies. In *ISWC*, 2010.

[40] G. Ladwig and T. Tran. SIHJoin: Querying remote and local Linked Data. In *ESWC*, 2011.

[41] A. Langegger, W. Wöß, and M. Blöchl. A Semantic Web middleware for virtual data integration on the Web. In *ESWC*, pages 493–507, 2008.

[42] Y. Li and J. Heflin. Using reformulation trees to optimize queries over distributed heterogeneous sources. In *ISWC*, 2010.

[43] M. Morsey, J. Lehmann, S. Auer, and A.-c. N. Ngomo. DBpedia SPARQL benchmark – performance assessment with real queries on real data. *ISWC*, 2011.

[44] S. Muñoz, J. Pérez, and C. Gutierrez. Simple and efficient minimal RDFS. *JWS*, 2009.

[45] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open Linked Data. *IJMSO*, 2008.

[46] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *ISWC*, pages 30–43, 2006.

[47] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[48] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, Jan. 2008.

[49] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, pages 524–538, 2008.

[50] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A benchmark suite for federated semantic data query processing. In *ISWC*, 2011.

[51] M. Schmidt, T. Hornung, N. Kuchlin, G. Lausen, and C. Pinkel. An experimental comparison of RDF data management approaches in a SPARQL benchmark scenario. *ISWC*, pages 82–97, 2010.

[52] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^{2}Bench: A SPARQL Performance Benchmark. In *ICDE*, pages 222–233, 2009.

[53] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on Linked Data. In *ISWC*, 2011.

[54] H. Stuckenschmidt, R. Vdovjak, G.-J. Houben, and J. Broekstra. Index structures and algorithms for querying distributed RDF repositories. In *WWW*, 2004.

[55] T. Tran, L. Zhang, and R. Studer. Summary models for routing keywords to Linked Data sources. In *ISWC*, 2010.

[56] J. Umbrich. A Hybrid Framework for Querying Linked Data Dynamically. NUI Galway, PhD Thesis. `http://aran.library.nuigalway.ie/xmlui/handle/10379/3360`, 2012.

[57] J. Umbrich, C. Gutierrez, A. Hogan, M. Karnstedt, and J. X. Parreira. The ACE Theorem for Querying the Web of Data. In *WWW Companion Volume*, 2013.

[58] J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Improving the recall of live linked data querying through reasoning. In *RR*, pages 188–204, 2012.

[59] J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over Linked Data. *WWWJ*, 2011.

[60] J. Umbrich, M. Karnstedt, A. Hogan, and J. X. Parreira. Hybrid SPARQL queries: Fresh vs. fast results. In *ISWC*, pages 608–624, 2012.

[61] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using MapReduce. In *ISWC*, pages 634–649, 2009.

[62] J. Weaver and J. A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In *ISWC*, pages 682–697, 2009.

Appendix

A. FedBench Queries

Herein, we present the results for the individual FedBench queries. We run the queries four times for each of the ten LiDaQ experiments, and for comparability across different configurations, we present the best run in terms of results returned, and if tied, by time; we thus select the run which provided the most stable behaviour and returned the most results. The variation between the four runs has already been analysed in Section 8.4. We also show results for the SQUIN library (v.0.1.3): we highlight that we only run the SQUIN implementation once since (to the best of our knowledge) it does not implement politeness policies, and thus the LiDaQ configurations may have an advantage in comparison—in any case, we show that SQUIN is generally faster in total query time than LiDaQ (likely due to shorter politeness delays) but slower for first-result times. We do not have measurements for the intermediate triples processed by SQUIN.

To avoid repetition, we discuss results incrementally; we may only briefly remark again on observations that have already been made for earlier queries.

LD1: LIST AUTHOR(S) WITH THEIR PAPER(S) FOR THE POSTER/DEMO TRACK OF ISWC 2008.

```
SELECT DISTINCT * WHERE {
  ?paper swc:isPartOf swIswc08pd: .
  ?paper swrc:author ?p .
  ?p rdfs:label ?n . }
```

Table 11
Benchmark results for query LD1

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
CORE	582	333	342.9	4.9	633	23,968	—
CORE$^-$	582	333	343.5	3.7	628	23,013	—
SEEALSO	582	333	361.5	3.6	704	25,229	—
SAMEAS	668	529	391.4	3.9	761	26,269	8,109
RDFS$_s$	615	380	478.8	3.8	628	23,013	11,501
RDFS$_d$	615	380	350.3	5.2	666	23,013	13,984
RDFS$_e$	615	380	356.1	6.6	865	23,013	18,587
COMB$_s$	715	692	571.9	4.4	842	29,212	26,804
COMB$_d$	713	680	461	8.4	1,002	28,485	27,745
COMB$_e$	715	692	512.6	15.8	1,269	29,212	35,418
SQUIN	582	333	86.8	28	703	—	—

The results for this query come mostly from one site: the `data.semanticweb.org` "Dog Food" server. The query engine first finds the list of URIs for all 85 demo/poster papers published at ISWC 2008 on the first document, dereferences these 85 URIs and builds a list of 288 unique authors, then finally dereferences these to find a list of 333 unique names (some authors have multiple versions of names). For CORE$^-$, the politeness policy of two lookups per second leads to query times of over 5 minutes (the fastest query time possible here is $\frac{\mathbf{HTTP}}{2}$ seconds). Conversely, although SQUIN performs more lookups than CORE$^-$ and CORE and generates the same results, it is much faster, but performs at least 8 HTTP lookups/second to `data.semanticweb.org` which is four times more than the bounds of our politeness policy. However, first results take 4 seconds for LiDaQ (or up to 16 seconds for dynamic schema) versus 28 seconds for SQUIN.

In this case, we see that CORE$^-$ saves few lookups and little time when compared with CORE, and that SEEALSO increases the number of sources but not the number of results. We see that RDFS reasoning finds some additional results: `foaf:name` and `skos:prefLabel` are found to be sub-properties of `rdfs:label` and provide additional name variations, including with language tags. Some of the authors have `owl:sameAs` relations to external sources, which, with SAMEAS, provide additional URIs for authors and name variations using a sub-property of label. The most results are thus given by the COMB approaches, which are also the slowest overall.[21]

LD2: LIST AUTHOR(S) WITH THEIR PAPER(S) IN PROCEEDINGS RELATED TO ESWC 2010.

```
SELECT DISTINCT * WHERE {
  ?proceedings swc:relatedToEvent swEswc10: .
  ?paper swc:isPartOf ?proceedings .
  ?paper swrc:author ?p . }
```

Although LD2 is very similar to LD1—requiring data mostly from the same Dog-Food provider—the measures in Table 12 show result sizes that are the same for all configurations: none of the extensions find any additional results, where RDFS$_s$ adds significant overhead. First results are often achieved in 4 seconds, unless schema data need to be collected. The source-selection savings for CORE$^-$ vs. CORE are notable, where CORE does not dereference the 173 authors bound to `?p` (requiring $173 \times 2 = 346$ lookups including 303 redirects) since `?p` bindings are not part of a join. We note that SQUIN performs fewer lookups than

[21]The additional answers available for RDFS and same-as reasoning can be seen from, *e.g.*, `http://data.semanticweb.org/person/mathieu-daquin/rdf`.

Table 12
Benchmark results for query LD2

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
CORE	236	185	260.3	3.9	478	20,356	—
CORE$^-$	236	185	69.8	3.5	128	3,662	—
SEEALSO	236	185	70	3.5	128	3,662	—
SAMEAS	236	185	70.3	3.6	128	3,662	—
RDFS$_s$	236	185	202.5	4	128	3,662	2,139
RDFS$_d$	236	185	73.6	5.7	148	3,662	8,193
RDFS$_e$	236	185	77.7	7	363	3,662	12,312
COMB$_s$	236	185	219	4.8	128	3,662	2,139
COMB$_d$	236	185	76.9	12.8	148	3,662	8,124
COMB$_e$	236	185	79.7	22.9	363	3,662	12,162
SQUIN	236	185	24	4.4	171	—	—

we would expect if it were to dereference authors, but still dereferences more URIs than CORE$^-$ and its analogues. As such, it would seem that SQUIN also implements some reduced source-selection optimisations.

LD3: LIST THE AUTHOR(S) WITH THEIR SAME-AS RELATION(S), AND WITH THEIR PAPER(S) FOR THE POSTER/DEMO TRACK OF ISWC 2008.

```
SELECT DISTINCT * WHERE {
  ?paper swc:isPartOf swIswc08pd: .
  ?paper swrc:author ?p .
  ?p owl:sameAs ?x ; rdfs:label ?n . }
```

Table 13
Benchmark results for query LD3

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
CORE	247	191	388.1	4	760	27,538	—
CORE$^-$	247	191	342.8	8.1	628	23,013	—
SEEALSO	247	191	360	8.2	704	25,229	—
SAMEAS	394	951	389.5	4.2	763	26,248	8,014
RDFS$_s$	263	246	474.7	5	628	23,013	11,501
RDFS$_d$	263	246	349.5	4.9	666	23,013	13,991
RDFS$_e$	263	246	355.8	4.8	865	23,013	18,775
COMB$_s$	422	1,469	569.3	10.4	839	28,485	25,797
COMB$_d$	425	1,583	461.8	18.8	1,008	29,212	28,814
COMB$_e$	425	1,583	511.6	19.1	1,269	29,212	35,547
SQUIN	247	191	87.3	32.2	728	—	—

This query adds a triple pattern to query LD1, restricting the list of authors to (explicitly) look for those with an `owl:sameAs` relation. This reduces the number of authors involved from 288 in LD1 to 54 in LD3. We can see in Table 13 that for configurations without reasoning, LD3 returns ~57% of the number of results of LD1: the decrease in authors is partially balanced by the addition of another variable in the results. CORE$^-$ offers a moderate performance improvement over CORE while returning the same results. RDFS reasoning increases result sizes for similar reasons as for LD1, and at little cost. SAMEAS shows a marked increase in results size: the additional `?x` variable is replaced by all equivalent URIs for each author, leading to an additional product of result terms.

LD4: LIST THE AUTHOR(S) WITH PAPER(S) IN THE PROCEEDINGS OF ESWC 2010 WHO ALSO HAD ROLE(S) AT THE CONFERENCE

```
SELECT DISTINCT * WHERE {
  ?role swc:isRoleAt swEswc10: .
  ?role swc:heldBy ?p .
  ?proceedings swc:relatedToEvent swEswc10: .
  ?paper swrc:author ?p .
  ?paper swc:isPartOf ?proceedings . }
```

Table 14
Benchmark results for query LD4

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
CORE	60	50	986.9	33.5	1,805	74,635	—
CORE$^-$	60	50	984.5	50.9	1,801	73,767	—
SEEALSO	60	50	1,019.4	51.9	1,982	81,864	—
SAMEAS	105	146	1,167.4	56.5	2,462	102,286	104,431
RDFS$_s$	60	50	1,140.2	17.5	1,801	73,767	45,352
RDFS$_d$	60	50	1,003	66	1,843	73,767	45,943
RDFS$_e$	60	50	1,023.2	11.5	2,173	73,767	58,374
COMB$_s$	162	203	4,658.4	68.9	2,834	115,707	297,620
COMB$_d$	162	203	2,249.4	172.4	3,383	115,663	557,880
COMB$_e$	80	126	7,211.6	52.9	9,225	109,858	1,702,602
SQUIN	60	50	244.3	236.7	1,981	—	—

Again, this query is an extension of LD2 and restricts the list of authors to those who, as well as having a paper at ESWC 2010, also had a role at the conference. Looking at the results in Table 14, even for CORE$^-$, the query processor performed over 1,800 lookups and our reduced source selection approach barely affects the number of lookups (in this case, `?p` falls into a join position and 251 people had a role at ISWC). The fastest time was around 16 minutes for CORE$^-$ (again, $\sim \frac{1{,}800}{2}$ seconds). However, the first results often arrived before the one minute mark. RDFS reasoning alone produces no additional results, but also does not overly influence runtime. Conversely, SAMEAS produces additional results, where author pages are this time dereferenced and `owl:sameAs` relations found, adding aliases for bindings in `?p`. The combined approaches became unstable, adding additional HTTP load to what is already a demanding query. In particular, COMB$_s$ and COMB$_e$ actually timeout after roughly two hours; for example, the COMB$_e$ approach retrieved almost ten thousand sources before timing out, where the `owl:sameAs` links from authors on `data.semanticweb.org` form a bridge to DBpedia, whose schema data has a high fan-out. From previous queries, we have seen that the schema data directly referenced from `data.semanticweb.org` is relatively easy to retrieve using dynamic import mechanisms; however, the schemata for other sites requires many more sources to retrieve, particularly in the RDFS$_e$/Comb$_e$ configurations.

LD5: List the name(s) of the album(s) by Michael Jackson

```
SELECT DISTINCT * WHERE {
  ?a dbowl:artist dbpedia:Michael_Jackson .
  ?a rdf:type dbowl:Album .
  ?a foaf:name ?n . }
```

Table 15
Benchmark results for query LD5

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
Core	85	43	63.3	6	121	9,017	—
Core$^-$	85	43	66.8	9	116	8,285	—
seeAlso	85	43	65.2	7.2	116	8,285	—
sameAs	313	271	212.7	14	593	15,179	119,907
RDFS$_s$	85	43	218.3	23.5	116	8,284	7,115
RDFS$_d$	83	42	417.7	9.4	698	8,203	163,163
RDFS$_e$	36	18	4,886.6	12.9	9,729	4,087	302,609
Comb$_s$	0	0	7,341.5	—	780	17,027	745,534
Comb$_d$	15	14	7,200.6	353.7	1,452	14,450	958,611
Comb$_e$	—	—	—	—	—	—	—
Squin	85	43	16.2	3.9	115	—	—

This query shifts the focus to the `dbpedia.org` data provider. First `dbpedia:Michael_Jackson` is dereferenced to retrieve URIs for Michael Jackson's albums, which are subsequently dereferenced to confirm that they are albums and to retrieve their name. Primarily, the results show that following `owl:sameAs` links from the DBpedia domain introduces high overhead: there are a total of 425 URI aliases for Michael Jackson and his albums on the DBpedia, including `owl:sameAs` links to `freebase.com`, `sw.cyc.com`, `linkedmdb.org` and `zitgist.com`. The fastest query times take about a minute for LiDaQ and 16 seconds for SQUIN.

Although sameAs runs through (taking $3.28\times$ longer than Core$^-$), when same-as and RDFS reasoning are combined, LiDaQ becomes unstable: all Comb approaches timed out, where Comb$_e$ threw an `OutOfMemoryException` in all four runs before the timeout was reached due to massive amounts of inferences. Furthermore, the RDFS$_e$ configuration without `owl:sameAs` reasoning showed that the dynamic import of extended schema does not work well for DBpedia, again touching upon nearly ten thousand sources and generating fewer results than Core$^-$ (which it extends). In general, the high fan-out of `owl:sameAs` and schema-level links on DBpedia—and on sites linked by DBpedia such as `sw.cyc.com`—combined with a query that already accesses over one hundred DBpedia pages in the baseline setup, prove too much for RDFS$_e$ and Comb approaches.

LD6: List the movie director(s) from Italy, their film(s) and the official name(s) of location(s) for the film(s)

```
SELECT DISTINCT * WHERE {
  ?director dbowl:nationality dbpedia:Italy .
  ?film dbowl:director ?director.
  ?x owl:sameAs ?film .
  ?x foaf:based_near ?y .
  ?y geo:officialName ?n . }
```

Table 16
Benchmark results for query LD6

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
Core	0	0	9.6	—	12	17,864	—
Core$^-$	0	0	6.8	—	2	10,001	—
seeAlso	0	0	3.6	—	2	10,001	—
sameAs	0	0	49.5	—	7	10,067	20,090
RDFS$_s$	0	0	145.2	—	2	10,001	4,580
RDFS$_d$	0	0	27.3	—	49	10,001	1,329
Comb$_s$	0	0	215.3	—	7	10,067	24,922
Comb$_d$	0	0	59	—	64	10,067	71,560
Comb$_e$	0	0	7,223.5	—	945	10,067	427,421
Squin	0	0	5.9	—	1	—	—

This query intends to span the DBpedia (first three patterns), LinkedMDB (fourth pattern) and GeoNames (fifth pattern) data providers. However, as we can see in Table 16, no setup returned any results. At the time of running the experiments, the dereferenced document for `dbpedia:Italy` contained 10,001 triples due to a manual cut-off set for the exporter, where many triples (including inlinks) were omitted and where the dereferenced document included no `dbowl:nationality` triples. At the time of writing, the dereferenced document contains 44,421 triples, including 842 `dbowl:nationality` inlinks. In any case, as we discuss for the next query, the GeoNames exporter hosting data for the final triple pattern bans access from all agents through its `robots.txt`. Aside from such issues, we would expect this query to pose a major challenge to LTBQE, and to again introduce unstable behaviour for Comb configurations.

LD7: List the name(s) of the parent feature(s) of Germany

```
SELECT DISTINCT * WHERE {
  ?x geo:parentFeature
          <http://sws.geonames.org/2921044/> .
  ?x geo:name ?n . }
```

LiDaQ will not run this query since the `robots.txt`[22] forbids software agents to access information on the `sws.geonames.org` domain. SQUIN does access the `sws.geonames.org` domain, but even aside from the `robots.txt` issue, the first query pattern is not matched by any data in the document dereferenced by the given GeoNames URI for Germany: dereferenced documents on `sws.geonames.org` do not contain triples where the URI appears in the object position.

LD8: LIST THE DRUG(S) IN THE MICRONUTRIENT CATEGORY, THEIR CAS REGISTRY NUMBER(S), ALIAS(ES), NAME(S) AND SUBJECT(S)

```
SELECT DISTINCT * WHERE {
   ?drug drugbank:drugCategory
                        drugbank:micronutrient .
   ?drug drugbank:casRegistryNumber ?id .
   ?drug owl:sameAs ?s .
   ?s foaf:name ?o .
   ?s dcterms:subject ?sub . }
```

Table 17

Benchmark results for query LD8

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
Core	39	19	78.9	17.6	351	20,655	—
Core$^-$	39	19	61.9	25.5	257	7,245	—
seeAlso	39	19	96.2	21.4	334	7,309	—
sameAs	294	21,071	1,374.4	67.1	856	12,839	515,297
RDFS$_s$	39	19	198.5	15.9	257	7,245	4,979
RDFS$_d$	8	4	181.4	132.6	231	774	43,814
RDFS$_e$	24	12	7,514.8	155.5	7,175	5,491	143,236
Comb$_s$	347	29,139	7,354.9	35.5	1,217	15,209	407,741
Comb$_d$	0	0	7,212.1	—	1,289	11,416	73,304
Comb$_e$	—	—	—	—	—	—	—
Squin	22	10	120.9	10.5	482	—	—

From the results in Table 17, we see the improvements of Core vs. Core$^-$. In a reverse of previous trends, SQUIN is faster to begin streaming results but slower to terminate than Core/Core$^-$. The results for this query show highly unstable behaviour for all reasoning extensions except RDFS$_s$. In particular, the consideration of `owl:sameAs` links snowballs and introduces huge amounts of inferences, which we believe to be due to data quality issues with this relation within Linked Drug Data, and which we had previously observed in other work [36, § 4.4].[23] This of course highlights the problem whereby—even with counter-measures such as authoritative analysis of schema data—reasoning exacerbates data quality issues for remote data providers. When `owl:sameAs` and dynamic RDFS import and reasoning is combined for Comb$_d$ and Comb$_e$, we encountered further `OutOfMemoryException`s.

LD9: LIST THE FOOTBALL TEAM(S) THAT WON A FIFA WORLD CUP AND THAT WERE MANAGED BY "LUIZ FELIPE SCOLARI".

```
SELECT DISTINCT * WHERE {
  ?x dcterms:subject
       dbpcat:FIFA_World_Cup-winning_countries .
  ?p dbpowl:managerClub ?x .
  ?p foaf:name "Luiz Felipe Scolari"@en . }
```

Table 18

Benchmark results for query LD9

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
Core	0	0	147.3	—	266	29,326	—
Core$^-$	0	0	147.4	—	260	27,821	—
seeAlso	0	0	136.4	—	260	27,821	—
sameAs	0	0	182.6	—	337	7,915	92,485
RDFS$_s$	0	0	299.2	—	202	22,791	19,642
RDFS$_d$	0	0	904.8	—	1,512	19,133	227,663
RDFS$_e$	0	0	1,607.5	—	3,488	4,928	33,810
Comb$_s$	0	0	7,342.9	—	984	28,211	558,187
Comb$_d$	0	0	7,202	—	1,437	16,109	1,432,297
Comb$_e$	—	—	—	—	—	—	—
Squin	0	0	25.6	—	247	—	—

As we can see from the results in Table 18, none of the setups returned any content. At the time of the experiments, the document for the Brazilian national football team (the answer to `?p`) contained parser errors (tested with the W3C validator), which have since been fixed. We again see that following `owl:sameAs` and schema level links from the DBpedia causes huge overheads, with Comb$_s$ and Comb$_d$ hitting timeouts, and Comb$_e$ again throwing an `OutOfMemoryException` after inferring too much data.

LD10: LIST THE CHANCELLOR(S) OF GERMANY, THEIR ALIAS(ES) AND LATEST ARTICLE(S)

[22]`http://sws.geonames.org/robots.txt`

[23]We refer the reader to `https://groups.google.com/forum/?fromgroups#!topic/pedantic-web/rXQPcFLMOi0` for detailed discussion.

```
SELECT DISTINCT * WHERE {
 ?n dcterms:subject
                 dbpcat:Chancellors_of_Germany .
 ?n owl:sameAs ?p2 .
 ?p2 nytimes:latest_use ?u . }
```

Table 19

Benchmark results for query LD10

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
CORE	0	0	55.5	—	165	15,008	—
CORE⁻	0	0	52.7	—	160	13,692	—
SEEALSO	0	0	52.6	—	160	13,692	—
SAMEAS	200	5,825	7,200.6	310.4	937	18,120	811,104
RDFS$_s$	0	0	195	—	160	13,692	17,008
RDFS$_d$	0	0	321	—	855	4,011	79,345
COMB$_s$	0	0	7,351	—	897	18,404	387,428
COMB$_d$	—	—	—	—	—	—	—
COMB$_e$	—	—	—	—	—	—	—
SQUIN	0	0	50.1	—	158	—	—

This query tries to combine the `dbpedia.org` and `data.nytimes.com` domains. As discussed in Section 8.4, we mapped `skos:subject` to `dcterms:subject` to reflect current DBpedia data. However, we found that although the content returned for the entities that are in the DBpedia category "Chancellors of Germany" contains several `owl:sameAs` relations to aliases in the `data.nytimes.com` domain, these are found in the inverse direction of the query pattern. This fact is reflected in the results shown in Table 19, where we only find results if `owl:sameAs` inferencing is enabled; in fact, both configurations that returned results timed out doing so, and again, both configurations involving the dynamic import of schema data threw exceptions.

LD11: LIST THE NAME(S) OF THE PLAYER(S) ON THE EINTRACHT FRANKFURT TEAM, THEIR BIRTHDAY(S) AND THE NAME(S) OF THEIR BIRTHPLACE(S)

```
SELECT DISTINCT * WHERE {
  ?x dbpowl:team dbpedia:Eintracht_Frankfurt .
  ?x rdfs:label ?y .
  ?x dbpowl:birthDate ?d .
  ?x dbpowl:birthPlace ?p .
  ?p rdfs:label ?l . }
```

Table 20 shows that this query involves the largest amount of results and source lookups of all the FedBench queries. The combination of over 300 players, each of which typically has labels in several languages and has two or three birth-places, each of which in turn has labels in several languages, leads to large results sets, even without reasoning. The number of HTTP

Table 20

Benchmark results for query LD11

Setup	Terms	Results	Time (s)	First (s)	HTTP	Data	Inferred
CORE	4,240	25,445	607.3	15.1	1,125	354,880	—
CORE⁻	3,936	23,621	595.8	7.6	1,073	336,615	—
SEEALSO	4,194	25,068	599.2	12.7	1,113	353,961	—
SAMEAS	40	572	7,210.2	80.9	3,741	94,263	2,327,965
RDFS$_s$	1,496	6,982	8,297.4	25.6	450	128,281	103,769
RDFS$_d$	1,281	7,319	2,392.5	27.7	3,896	123,839	271,772
RDFS$_e$	—	—	—	—	—	—	—
COMB$_s$	259	77,484	7,345.9	104.5	3,077	97,925	575,314
COMB$_d$	275	196,448	7,201.6	51	5,974	92,285	1,577,530
COMB$_e$	240	157,198	7,207.9	92.2	17,996	21,574	1,930,660
SQUIN	2,673	15,900	158.9	31.1	1,116	—	—

lookups also reflects the breadth of this query, primarily due to lookups on players and places. The reasoning extensions again exhibit unstable behaviour, either eventually timing-out or throwing an exception.

B. QWalk Results

Table 21 presents the detailed average results for the QWalk experiments across all query classes. For space reasons, we only present standard deviations for terms, results and time.

C. CURIE Prefixes

The CURIE prefixes used in this paper are enumerated in Table 22.

Table 22

Mappings for all prefixes used

Prefix	URI
cb:	`http://www.bizer.de#`
cbDoc:	`http://www4.wiwiss.fu-berlin.de/bizer/foaf.rdf`
dblp:	`http://dblp.l3s.de/d2r/`
dblpA:	`http://dblp.l3s.de/d2r/resource/authors/`
dblpADoc:	`http://dblp.l3s.de/d2r/data/authors/`
dblpP:	`http://dblp.l3s.de/d2r/resource/publications/conf/semweb/`
dblpPDoc:	`http://dblp.l3s.de/d2r/data/publications/conf/semweb/`
dbpcat:	`http://dbpedia.org/resource/Category:`
dbpedia:	`http://dbpedia.org/resource/`
dbpprop:	`http://dbpedia.org/property/`
dbpowl:	`http://dbpedia.org/ontology/`
dcterms:	`http://purl.org/dc/terms/`
drugbank:	`http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/`
ebiz:	`http://www.ebusiness-unibw.org/ontologies/consumerelectronics/v1`
foaf:	`http://xmlns.com/foaf/0.1/`
geo:	`http://www.geonames.org/ontology#`
nytimes:	`http://data.nytimes.com/elements/`
oh:	`http://www.informatik.hu-berlin.de/~hartig/foaf.rdf#`
ohDoc:	`http://www.informatik.hu-berlin.de/~hartig/foaf.rdf`
owl:	`http://www.w3.org/2002/07/owl#`
rdf:	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs:	`http://www.w3.org/2000/01/rdf-schema#`
skos:	`http://www.w3.org/2004/02/skos/core#`
swc:	`http://data.semanticweb.org/ns/swc/ontology#`
swrc:	`http://swrc.ontoware.org/ontology#`
swIswc08pd:	`http://data.semanticweb.org/conference/iswc/2008/poster_demo_proceedings`
swEswc10:	`http://data.semanticweb.org/conference/eswc/2010`

Table 21
Detailed QWalk results for all query classes

	Setup	**Term**		**Results**		**Time** (s)		**First** (s)	**HTTP**	**Data**	**Inferred**
		avg.	σ	*avg.*	σ	*avg.*	σ				
edge-s	Core	19.35	±37.75	16.78	±37.94	16.79	±8.19	6.6	17.78	8,676.43	—
	Core$^-$	19.33	±37.72	16.77	±37.91	9.99	±4.81	6.6	2.92	5,772.5	—
	seeAlso	19.33	±37.72	16.77	±37.91	10.04	±5.05	6.42	3.15	5,778.68	—
	sameAs	25.28	±63.42	22.73	±63.76	10.57	±5.47	6.93	3.27	5,579.42	67.82
	RDFS$_S$	24.13	±37.95	21.77	±38.01	16.93	±35.6	6.57	4.55	22,591.2	16,328.55
	Comb$_S$	30.07	±63.66	27.73	±63.91	18.08	±38.58	9.5	5.07	22,424.57	16,423.52
edge-o	Core	54.16	±382.48	53.53	±382.42	13.49	±8.5	6.19	7.18	3,517.61	—
	Core$^-$	54.18	±382.47	53.51	±382.42	8.25	±5.72	6.17	2.61	1,284.07	—
	seeAlso	54.19	±382.47	53.53	±382.42	8.69	±5.65	6.16	2.77	1,832.04	—
	sameAs	55.04	±382.37	54.35	±382.32	9.53	±8.07	6.08	3.37	1,984.7	171.67
	RDFS$_S$	54.28	±382.46	54.04	±382.38	10.06	±13.42	6.12	2.61	4,557.19	2,789.54
	Comb$_S$	55.14	±382.35	54.88	±382.27	12.52	±17.02	6.37	3.46	5,043.68	3,171.05
edge-so	Core	16.32	±15	35.34	±51.01	17.49	±8.08	6.65	19.86	3,853.07	—
	Core$^-$	16.14	±15.04	34.66	±49.83	12.28	±6.41	6.79	6.46	1,296.02	—
	seeAlso	16.17	±15.02	34.68	±49.82	13.01	±7.22	6.65	7.49	2,551.37	—
	sameAs	18.19	±17.4	56.14	±93.8	15.15	±11.48	6.68	8.53	1,776.64	393.93
	RDFS$_S$	19.71	±17.52	52.86	±73.08	14.19	±14.12	6.65	8.64	4,600.24	2,833.88
	Comb$_S$	21.78	±19.66	81.22	±122.66	16.84	±17.08	6.7	11.32	6,317.88	4,121
star-0-3	Core	3.49	±4.46	2.64	±4.4	14.9	±16.5	7.43	13.76	2,099.61	—
	Core$^-$	3.49	±4.46	2.64	±4.4	7.47	±2.57	6.95	1.9	595.42	—
	seeAlso	3.49	±4.46	2.64	±4.4	7.94	±4.64	7.43	1.91	595.42	—
	sameAs	4.03	±6.4	8.76	±49.72	9.31	±6.82	7.44	2.31	3,098.49	54.58
	RDFS$_S$	3.49	±4.46	2.64	±4.4	10.88	±25.86	7.24	1.9	10,328.1	6,816.55
	Comb$_S$	4.03	±6.4	8.76	±49.72	12.83	±31.08	7.78	2.33	9,920.43	6,876.52
star-1-2	Core	9.05	±31.78	11.35	±53.6	65.84	±391.73	7.25	13.26	1,709.11	—
	Core$^-$	9.05	±31.78	11.35	±53.6	7.08	±2.18	6.72	1.82	48.74	—
	seeAlso	9.08	±31.79	11.68	±53.8	8.68	±10.44	6.65	2.21	62.81	—
	sameAs	9.53	±31.83	13.02	±54.21	8.15	±3.94	6.71	2.4	323.32	52.97
	RDFS$_S$	12.56	±53.7	644	±5,028.56	9.07	±12.49	6.88	1.82	856.34	556.48
	Comb$_S$	13.06	±53.7	645.95	±5,028.32	10.98	±17.91	6.68	2.74	1,354.61	1,017.95
star-2-1	Core	5.53	±12.82	7.44	±30.09	12.77	±16.11	6.41	11.79	515.37	—
	Core$^-$	5.51	±12.82	7.43	±30.09	6.63	±2.13	6.25	1.73	104.11	—
	seeAlso	5.51	±12.82	7.43	±30.09	7.46	±3.37	6.23	1.93	104.5	—
	sameAs	5.66	±12.8	7.57	±30.07	7.25	±2.69	6.57	1.79	372.14	14.16
	RDFS$_S$	6.1	±13.5	17.04	±87.46	7.98	±10.98	6.31	1.73	814.74	409
	Comb$_S$	6.24	±13.47	17.19	±87.44	9.84	±13.66	6.99	1.99	797.06	435.4
star-3-0	Core	2.2	±0.79	1.15	±0.56	10.18	±5.65	7.1	6.79	1,242.77	—
	Core$^-$	2.2	±0.79	1.15	±0.56	7.16	±3.26	6.69	1.53	949.88	—
	seeAlso	2.2	±0.79	1.15	±0.56	7.76	±3.99	6.75	1.76	964.27	—
	sameAs	2.29	±1.15	1.24	±1.01	8.33	±5.6	6.93	1.83	2,138.38	68.39
	RDFS$_S$	3.67	±2.57	2.7	±2.61	11.46	±26.25	7.51	1.53	8,817.83	6,742.48
	Comb$_S$	3.77	±2.66	2.8	±2.7	13.8	±36.07	9.58	1.95	9,411.11	7,203.71
s-path-2	Core	6.89	±38.25	6.35	±38.27	14.42	±34.39	7.05	14.02	575.45	—
	Core$^-$	6.89	±38.25	6.35	±38.27	8.86	±4.98	6.38	2.8	236.39	—
	seeAlso	6.89	±38.25	6.35	±38.27	9.38	±5.35	6.37	3.08	236.45	—
	sameAs	7.08	±38.24	6.74	±38.32	9.62	±5.74	6.25	3.52	757.2	71.17
	RDFS$_S$	7.79	±38.17	7.62	±38.28	9.14	±5.25	6.49	2.86	2,016.91	1,325.12
	Comb$_S$	8	±38.17	8.03	±38.32	10.25	±5.95	6.43	3.98	2,214.55	1,502.45
s-path-3	Core	48.14	±237.92	28.55	±122.05	16.21	±22.62	6.61	16.53	2,605.86	—
	Core$^-$	48.1	±237.89	28.43	±121.89	11.6	±6.73	6.89	4.59	1,302.96	—
	seeAlso	49.35	±237.82	29.06	±121.83	11.37	±8.02	6.51	5.94	1,305.53	—
	sameAs	48.14	±237.89	28.47	±121.88	11.59	±5.82	6.74	4.71	4,612.86	4.04
	RDFS$_S$	48.69	±237.78	29.02	±121.76	11.94	±6.84	7.1	4.61	13,502.16	8,601.94
	Comb$_S$	51.59	±238.09	30.49	±121.85	14.54	±17.97	6.88	7.76	13,329.9	8,657.35
o-path-2	Core	96.02	±379.55	94.26	±376.01	68.52	±262.1	7.34	107.06	2,942.29	—
	Core$^-$	96.02	±379.55	94.26	±376.01	11.23	±7.84	7.16	4.18	1,260.85	—
	seeAlso	96.02	±379.55	94.26	±376.01	12.15	±7.95	6.87	4.52	1,261.71	—
	sameAs	140.79	±566.69	139.19	±564.44	13.19	±9.95	7.24	6.21	7,333.55	1,146.18
	RDFS$_S$	96.08	±379.59	96.03	±377.18	15.09	±23.65	7.14	4.18	14,604.69	10,073.15
	Comb$_S$	140.87	±566.71	140.98	±565.07	20.47	±36.77	7.32	6.69	18,734.92	12,521.85
o-path-3	Core	83.49	±268.83	86.06	±288.12	90.78	±268.35	7.41	157.46	7,293.86	—
	Core$^-$	83.51	±268.87	86.09	±288.15	15.1	±8.86	7.9	7.43	1,105.51	—
	seeAlso	83.51	±268.87	86.09	±288.15	14.75	±8.35	7.51	7.49	1,105.66	—
	sameAs	83.51	±268.87	86.09	±288.15	14.16	±8.98	7.09	7.43	3,854.49	—
	RDFS$_S$	83.51	±268.87	86.09	±288.15	17.1	±15.34	7.14	7.43	9,336.74	5,481.91
	Comb$_S$	83.51	±268.87	86.09	±288.15	18.38	±18.07	8.01	7.51	9,332.71	5,477.31