On Link Traversal Querying for a diverse Web of Data

Jürgen Umbrich [a,*], Aidan Hogan [a], Axel Polleres [b] and Stefan Decker [a]
[a] *Digital Enterprise Research Institute, National University of Ireland, Galway; Ireland*
E-mail: {juergen.umbrich,aidan.hogan,stefan.decker}@deri.org
[b] *Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna; Austria*
E-mail: axel.polleres@siemens.com

Abstract. Traditional approaches for querying the Web of Data 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 help bridge data heterogeneity in 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 returned; 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]. Such publishing is governed by four *Linked Data Principles*, here paraphrasing Berners Lee [4]:

LDP1: *use URIs to name things*, such that

LDP2: those *URIs can be dereferenced via HTTP*, such that

LDP3: *dereferencing yields useful RDF content* about that which is named, such that

LDP4: the returned *content includes links* (mentions external URIs) for further discovery.

The resulting collective of interlinked contributions from a wide variety of publishers has been dubbed the "*Web of Data*": a novel corpus of structured data distributed across the entire Web, described using the Semantic Web standards and made available to all under the above Linked Data principles.

*Corresponding author. E-mail: juergen.umbrich@deri.org

On the Web of Data, the URIs that are used to name resources (often) map through HTTP to the physical location of structured data about them. As such, the Web of Data itself can be viewed as forming a scale-free, decentralised database, consisting of millions of Web documents exposing (or one could even say *indexing*) structured data [6, 32, 56]. Further still, thanks to the provision of typed "RDF links" between such documents [34, § 4.5], agents can traverse and navigate the Web of Data to discover related information in an *ad hoc* manner.

Given this rich collection of diverse structured data, an open challenge is how to query this Web of Data in an efficient and effective manner. SPARQL [49]—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 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 [62]. Thus, if a source is missing from the replicated index or if the remote version of the source has changed since index time, users of the centralised query engine will often encounter stale or even missing results [62].

As opposed to considering a monolithic RDF graph, SPARQL encodes the notion of *Named Graphs*, which (loosely) corresponds to a means of logically partitioning some RDF corpus, such that combinations of partitions can be queried in isolation. Often, these partitions are based on the provenance of data, with the named-graph URI corresponding to the location from which an RDF document is retrieved; *e.g.*, a Web location.[1] Thus, given a (HTTP) correspondence between graph names and addresses, SPARQL over the Web of Data could be supported by means of *live querying* such that the content of these graphs is retrieved from the Web and processed at runtime to generate answers. By circumventing the need for replicated indexes and instead accessing remote data *in situ*, live querying would no longer suffer problems with incomplete or stale data.

[1]More correctly, SPARQL operates over *Internationalized Resource Identifiers* (IRIs), but we stick with the more familiar notion of "URIs" for readability.

However, SPARQL semantics considers a fixed dataset from which to generate query answers, whereas live querying explores an *a priori* unbounded Web of Data for on-the-fly answers without ever considering the dataset it operates over in its entirety. Furthermore, SPARQL queries need not specify the graphs (using, *e.g.*, `FROM`, `FROM NAMED` or `GRAPH` clauses) over which it should be run; oftentimes, queries are run over a default graph informally considered to consist of a merge of all sources. Thus, given a SPARQL query without any explicit graphs mentioned, a live query engine would still need to automatically determine which sources on the Web of Data could be *query-relevant*.

For finding such sources, the live query engine could leverage Linked Data principles, which state that URIs should be dereferenceable and give *follow-your-nose* cues as to where RDF data about a given resource (mentioned in queries or in intermediate results) might be found on the Web of Data. As of yet, SPARQL does not formally leverage the former set of principles. However, this observation prompted Hartig *et al.* [29] to investigate using dereferenceable URIs in the query—and recursively, in the intermediate results of the query—to automatically determine a focused set of sources which, by Linked Data principles, are likely to help answer a SPARQL query. These query-relevant sources are then retrieved and used to generate answers to the user query, and possibly recursively, to traverse links and find further query relevant sources. When operating over sufficiently compliant Linked Data, their approach bypasses the need for source graphs to be replicated locally or to be explicitly named in the original SPARQL query and allows for new sources to be discovered in an *ad hoc* manner by traversing links at query time. Later work by Hartig [27] calls this particular live querying approach: "*Link Traversal Based Query Execution*" (which we abbreviate to *LTBQE*).

However, in the LTBQE approach, remote lookups can often taken seconds to yield content, many lookups may be required, and subsequent lookups to the same remote server may need to be artificially delayed to ensure "polite" and sustainable consumption (*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 remote sources (to keep response times low) while maximising the number of answers returned; the approach relies on Linked Data principles as cues to identify a minimal amount of sources that maximise results. Relatedly, the success of LTBQE is premised on the assumption that most (or ideally all) query relevant data about a resource can be found in its respective dereferenced document; as we show later, this assumption only partially holds.

Herein, we look at the recall of LTBQE in practice and propose extensions to find additional answers. In particular, we propose that lightweight reasoning extensions—specifically relating to `owl:sameAs` and RDFS semantics—can help to "'squeeze" additional answers from sources, to find additional query-relevant sources, and to generally bridge diversity between different data providers on the 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 exacerbate performance issues with complex queries.

Paper contributions and structure. This paper is an extended version of previous work [60] where we originally proposed and evaluated our reasoning extensions. Herein, we additionally propose different mechanisms for dynamically importing schema data, add two more benchmarks for testing LTBQE and its extensions, and greatly extend discussion, particularly in the context of related work and the performance of LTBQE in uncontrolled environments.

More concretely, this paper is 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, which emulates the iterator-based model of SQUIN [29], but also features novel reasoning extensions and optimisations.

§ **6** We analyse a crawl of ~ 7.4 m RDF/XML documents from the Web of Data (*viz.*, the BTC'11 dataset), looking at the ratio of triples returned in documents dereferenced to by some URI vs. all data available about that URI in the entire sample; we also look at how much more raw dereferenceable data our reasoning extensions make available to LTBQE.

§ **7** We give an overview of current SPARQL benchmarks that can be run against real-world Linked Data sources, and survey how related Linked Data querying papers evaluate their works. We propose a novel benchmark methodology called QWalk, which addresses shortcomings of existing benchmarks.

§ **8** We test LTBQE and its extensions for three query benchmarks in a realistic, uncontrolled setting, presenting detailed measures comparing performance, result sizes, sources accessed, and so forth.

§ **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 very comprehensive and detailed overview about the existing different approaches to query Linked Data was recently published by Hose *et al.* [37]. We similarly classify relevant query approaches into three categories:

1. materialised systems and data warehouses,
2. systems which 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. We focus throughout on the execution of SPARQL queries against the Web of Data.

With respect to the extensions that we propose for LTBQE, our work also relates to ongoing research into reasoning approaches that are tailored for execution over Web data. We thus also cover some background research and techniques in this area.

2.1. Materialised Systems

Materialised query-engines locally replicate the content of remote Web of Data sources in a quad store and execute SPARQL queries over the local copy. Such approaches typically feature a crawler or other data acquisition component which, *e.g.*, follows links between documents to discover new information, and/or downloads documents which have been requested for indexing by remote parties. The primary targets for such materialised engines are:

1. to have as much coverage of the Web of Data as possible,
2. to keep results up to date, and
3. to be able to process potentially expressive (*i.e.*, expensive) SPARQL queries in an efficient manner and with high concurrency.

These objectives are (partially) met using distribution techniques, replication, optimised indexes, compression techniques, data synchronisation, and so on [8,17, 25,46]. Still, given that such services often index millions of documents, they require large amounts of resources to run. In particular, maintaining a local, *up-to-date* index with good coverage of the Web of Data is a Sisyphean task.

In previous years, we supported such a service powered by YARS2 [25] allowing for querying over millions of RDF Web documents (and some of their entailments), but have since discontinued the endpoint due to prohibitive running and maintenance overheads (in a research setting). Current centralised SPARQL endpoints harvesting Linked Data include "FactForge" [7][2] (powered by BigOWLIM [8]), OpenLink's LOD cache[3] and Sindice's "Semantic Web Index" [46][4] (both powered by Virtuoso [17]).

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

[2]http://factforge.net/sparql
[3]http://lod.openlinksw.com/sparql
[4]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[5], *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. Some systems—such as SPARQL-DQP [3]—require that sub-queries are annotated with the SPARQL 1.1 `SERVICE` keyword, which allows users to invoke remote endpoints and, more generally, to state which parts should be routed where.

Other federated SPARQL engines locally index "service descriptions" or "catalogues", which describe the contents of remotes endpoints and are used to split and route sub-queries without explicit `SERVICE` annotations [50]. One of the earliest works going in this direction (and which predated SPARQL by over three years) was by Stuckenschmidt *et al.* [57], who proposed summarising the content of distributed RDF repositories using schema paths (non-empty property chains). The SemWIQ [42] architecture uses counts of the extension of each class and property in an endpoint to create a catalogue used for routing queries; later work extended the set of available statistics using a tool called RDFStats [41], which also provides histograms covering *e.g.*, subjects or data types, as well as estimated cardinalities of selected (sub-)queries. SPLENDID [21] is another federation infrastructure, which uses the Vocabulary of Interlinked Datasets (VoID) to describe the content of endpoints [1]. An alternative to pre-computed service descriptions—used by FedX [54] and SPLENDID—is to instead probe SPARQL endpoints with `ASK` sub-queries to see if they have relevant information during query-time; this information can be cached and re-used.

In recognition of the growing popularity of federated SPARQL, the SPARQL 1.1 W3C Working Group has added some new federation features [24]. As already mentioned, the `SERVICE` keyword can be used to invoke remote endpoints [24], and the new `VALUES` (previously known as `BINDINGS`) feature can be used to "ship" batches of intermediate bindings to an end-

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

point [24]. In addition, the "SPARQL 1.1 Service Description" proposes a vocabulary for describing the functionalities and datasets of SPARQL endpoints in a standard way [67].

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

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. Conceptually, Ladwig & Tran [39] identify three categories of such approaches: (i) top-down query evaluation, (ii) bottom-up query evaluation, and (iii) mixed strategy query evaluation.

Top-down evaluation determines the query relevant sources before the actual query execution using knowledge about the available sources stored in a so-called "source-selection index". These source-selection indexes can vary from simple inverted-index structures [43, 46], to query-routing indexes [59], schema-level indexes [57], and lightweight hash-based structures [61].

The *bottom-up* query evaluation strategy involves discovering query-relevant sources on-the-fly during the evaluation of queries. The LTBQE approach [29] and the work in the present paper fall into this category. A "seed set" of remote query-relevant sources are dereferenced from URIs mentioned in the query; links are followed from these initial sources to find further query relevant sources and to find more answers or satisfy additional sub-goals in the query; the process continues recursively until all known query-relevant sources have been exhausted. Since no local index is required, this approach can be used in decentralised scenarios, where clients can execute queries remotely over the Web without accessing a centralised service. The unique challenges for such an approach are (i) to find as many query-relevant sources as possible to improve recall of answers; (ii) to conversely minimise the amount of sources accessed to avoid traffic and slow query-response times; (iii) to optimise query execution in the absence of typical selectivity estimates, *etc.* [27, 29]. In this paper, we focus on the first two challenges. The theoretical foundation for LTBQE was later published by Bouquet *et al.* [12] and further developed by Hartig *et al.* [28, 30]. We will elaborate more upon the LTBQE approach in Section 4.

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]. There is still huge variance possible between the different approaches, targeting different scenarios and use-cases. For example, the traditional inverted index proposed by Sindice [46] is still very much a lightweight version of a centralised service. Conversely, our hash-based data summaries approach [61] is more geared towards lightweight, client-side processing. Depending on the particular approach taken, challenges may vary (as before) between those identified for top-down and bottom-up strategies; for example, keeping local knowledge up-to-date, or identifying a low number of query-relevant sources, *etc.*

2.4. Hybrid and Navigational Query Engines

Moving towards a mature Linked Data query-answering system, one could thus consider a combination of approaches, where each has its complementary advantages and disadvantages. An interesting and relatively novel research area would then be investigating how to combine local and remote querying techniques—both on a theoretical, engineering, and social level—to complement the fast but potentially stale results of a centralised engine with slower but fresher live results. A number of works have tackled this combination on a variety of levels (see, *e.g.*, [32, 40, 62]). We see a lot of promise in this direction: we believe that the real strength of live query approaches such as LTBQE lie in combination with other query paradigms. In fact, we have already begun to look at combining LTBQE with materialised approaches for answering SPARQL queries [62]. For example, using materialised or top-down approaches seems well suited for relatively static data (*e.g.*, DBpedia, DBLP, *etc.*), whereas bottom-up approaches seem better suited for dynamic data [63] (*e.g.*, identi.ca, MusicBrainz, *etc.*) or for accessing potentially sensitive remote data (which may require user-specific access control at runtime), with hybrid strategies required to effectively blend support for both.

In this light, various authors have also questioned whether SPARQL is the right language to query the

Web of Data: again, SPARQL is defined for closed datasets and was originally proposed with materialised settings in mind. Relatedly, there have been a number of proposals to extend SPARQL with regular expressions that capture navigational patterns, including work by Alkhateeb *et al.* [2] and work on the nSPARQL language [48].[6] Recently, Fionda *et al.* [18] proposed NautiLOD, a novel declarative language for navigating paths in the Web of Data guided by regular expressions over RDF predicates, using SPARQL `ASK` queries to *test* some conditions over the data encountered (*i.e.*, to find data matching a query), and allowing to trigger some actions whenever some condition is met. Such work goes beyond pure SPARQL querying, but perhaps touches upon some of the broader potential of 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. As we will show, these extensions help to compute additional answers from query-relevant sources on the Web of Data, as well as helping to discover additional sources. We now cover some related works in the area of reasoning over RDF Web data, which use the RDFS and OWL standards to integrate data from different sources based on the well-defined links provided by publishers.

For the moment, we wish to investigate a terse profile of reasoning which is useful for Web data. Along similar lines, Glimm *et al.* [20] surveyed the use of RDFS and OWL features in a large crawl of the Web of Data (*viz.*, BTC'11), applying PageRank over the documents contained within and summating the rank of all documents using each feature. They found that RDF(S) features were the most prominently used.[7] Out of the OWL features, they found that `owl:sameAs` occurred most frequently, though other features of OWL like `owl:FunctionalProperty` had a higher rank: the most broadly linked (and thus highly ranked) documents in the Web of Data are vocabularies, not the "data-level documents" in which `owl:sameAs` relations frequently appear.

With respect to scalable and efficient 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, 64, 66]. The core assumptions are that the volume of schema data is much smaller than instance data, that schema data are the most frequently accessed part of the knowledge-base during reasoning, that schema data are often 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.

Aside from pure efficiency and scalability concerns, the freedom associated with publishing on the Web—where anyone can say anything (almost) anywhere—causes significant obstacles with respect to the trustworthiness of data when performing 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 [11]. Various 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 [11, 13, 35]. We later use authoritative reasoning to avoid the unwanted effects of third-party schema contributions during RDFS reasoning. Delbru *et al.* [15] propose an alternative solution called *context-dependent reasoning* (or quarantined reasoning), where a closed scope is defined for each document being reasoned over, incorporating the document itself and (recursively) other documents it imports or links. Thus, obscure third party documents cannot inject unwanted schema into the inferencing process since they fall outside the quarantined scope. 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 au-

[6] SPARQL 1.1 includes a similar notion called *property paths* [24].

[7] Respectively from features ranked 1–6: `rdf:Property`, `rdfs:range`, `rdfs:domain`, `rdfs:subClassOf`, `rdfs:Class`, `rdfs:subPropertyOf`.

thors have looked specifically at the use and the quality of use of `owl:sameAs` on the Web of Data [16, 23, 36]. Halpin *et al.* [23] 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 found that judges often disagreed on what resources should (not) be considered the same. They estimated an accuracy for `owl:sameAs` links—where sameness could be confidently asserted for the sampled relations—at around 51% (±21%). These figures were much more pessimistic than a similar analysis that we conducted of one thousand `owl:sameAs` relations, where we asked a different question—is there anything between these two resources to confirm that they are not the same?—and where we estimated a respective precision of 97.2% [36].[8] We do not tackle issues relating to the quality of `owl:sameAs` relations in this work, but acknowledge this as a general and orthogonal challenge for Linked Data researchers to overcome [23, 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 techniques to additionally materialise inferences over remote data. In previous years, the SAOR reasoner [35] (used later) provided reasoning for the (now discontinued) YARS2 SPARQL index over RDF Web data. As already discussed, Sindice uses context-dependent reasoning to complement their local indexes [15]. Factforge incorporates reasoning techniques for the selected subsets of Linked Data that they index [7]. In terms of reasoning for live querying approaches, in their top-down system, Li and Heflin [43] also use reasoning techniques to perform query rewritings, as well as to generate inferences and find further results.

2.6. Novelty of Present Work

Having covered various aspects of background and related 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 look at 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 from inferences. 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 a bottom-up live query engine 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.[9] 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;

`dblpADoc:Olaf...`, `dblpADoc:Chris...` refer to information exported from the "DBLP Computer Science Bibliography"[10] 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

[8]This analysis was applied over pairs sampled from the *closure* of same-as relations. During our analysis, we found many pairs of resources for which very little knowledge was locally or externally available (for one or both). If there was no information to suggest that they were *not* the same, we would give this pair the "benefit of the doubt", taking the perspective that merging (aka. consolidating) the resources would not cause any notable data issues.

[9]As last accessed on 2012-06-23.

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

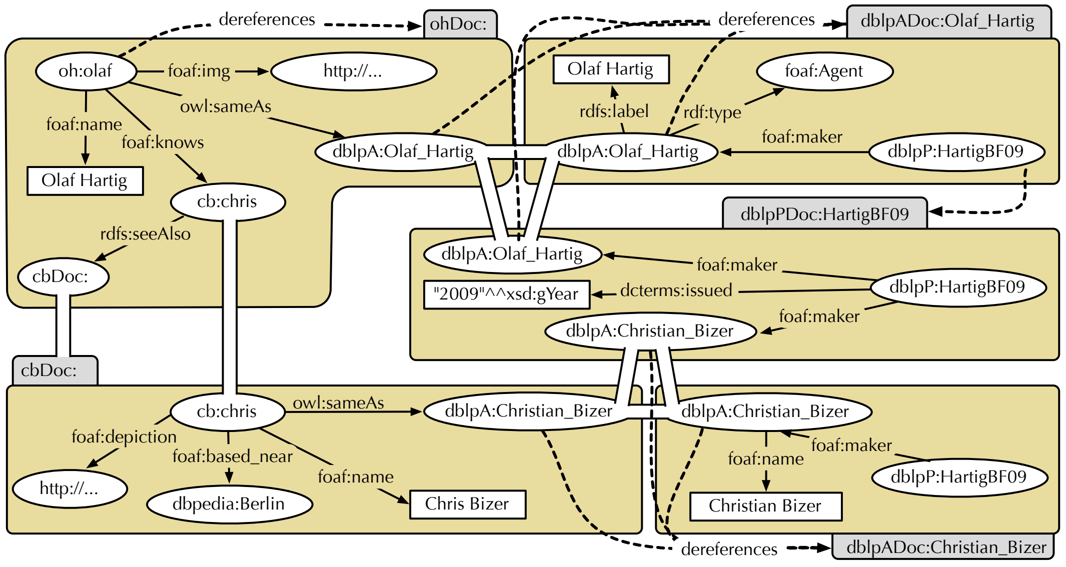

Fig. 1. Snapshot of a subgraph of five documents from the Web of Data. Individual documents are associated with individual background panes. The URI of each document is attached to its pane with a shaded tab. The same resources appearing in different documents are joined using "bridges". Links from URIs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

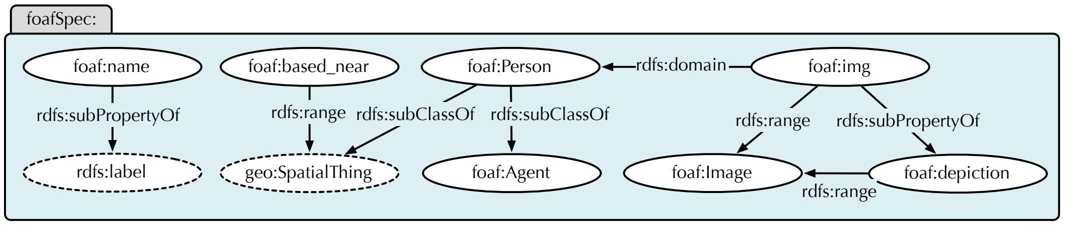

Fig. 2. Snapshot of an example schema document from the Web of Data, taken from the Friend Of A Friend (FOAF) Ontology. External terms are represented in ellipses with dashed lines.

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 we leave it 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].

3.2. RDF

In order to formally define our methods later, we must first provide some standard notation for dealing with RDF [33].

Definition 1 (RDF Term, Triple and Graph)**.** *The set of* RDF terms *consists of the set of URIs* $\mathbf{U}$*, the set of blank-nodes* $\mathbf{B}$ *and the set of literals* $\mathbf{L}$ *(which includes plain and datatype literals). 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 set-union). Here* s *is called subject,* p *predicate, and* o *object. A finite set of RDF triples* $G \subset \mathbf{G}$ *is called an* RDF graph*. We use the functions* $\mathsf{subj}(G)$*,* $\mathsf{pred}(G)$*,* $\mathsf{obj}(G)$*,* $\mathsf{terms}(G)$*, to denote the set of all terms projected from the subject,* *predicate, object and any position of a triple $t \in G$ respectively.*

3.3. Linked Data

As mentioned in the introduction, the four Linked Data principles [4] 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: *Include links by using external URIs for further discovery*

We now provide some notation which helps to formalise these principles and relate them to RDF. As per [29], we currently do not consider temporal issues with, *e.g.*, HTTP-level functions.

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* $\mathbf{S} := \{s \in \mathbf{U} : \mathsf{get}(s) \neq \emptyset\}$. *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 '$\uplus$' denotes the RDF merge of RDF graphs: a set union where blank nodes are rewritten to ensure that no two input graphs contain the same blank node label [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 issue a HTTP redirect to another URI with a `30x` response code, with the target URI listed in the `Location:` field of the HTTP header. We model this redirection 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 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* $\mathbf{D} := \{d \in \mathbf{U} : \mathsf{deref}(d) \neq \emptyset\}$*; note that* $\mathbf{S} \subset \mathbf{D}$ *and we place no expectations on what* $\mathsf{deref}(d)$ *returns, other than returning* some *valid RDF. As a shortcut, we denote by* $\mathsf{derefs} : 2^{\mathbf{U}} \to \mathbf{U} \times 2^{\mathbf{G}}$*;* $U \mapsto \{(\mathsf{redirs}(u), \mathsf{deref}(u)) \mid u \in U \cap \mathbf{D})\}$ *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).*[11]

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 [47,49]. 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 `OPTIONAL` in the original SPARQL specification and `MINUS`/(`NOT`) `EXISTS` defined in SPARQL 1.1 which assume a closed dataset—can be layered on

[11]We instantiate the formal model of Hartig [28] with 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.

top [47]. 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* $\mathbf{Q} := \mathbf{VUL} \times \mathbf{VU} \times \mathbf{VUL}$*. 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*.*

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. Again taking Γ from Figure 1, if we let Q be as follows:

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

Query 1: Authors and date of paper

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. We also give some definitions relating to rule-based inferencing, which we will use later. In particular, we support a small subset of OWL 2 RL/RDF rules, given in Table 1, which constitute a partial axiomatisation of the OWL RDF-Based Semantics. Our RDFS rules are the subset of the ρDF rules proposed by Muñoz *et al.* [45] that deal with instance data entailments (as opposed to schema-level entailments).[12] Our subset of OWL rules are specifically chosen to support the semantics of equality (particularly replacement) for `owl:sameAs`. Note that these rules support the RDFS/OWL features originally recommended for use by Bizer *et al.* when publishing Linked Data [9, §4.2, §6]. The rules we consider are given in Table 1. More recent guidelines [34, §4.4.3] recommend use of additional OWL features; we leave support for more expressive OWL reasoning to future work. For convenience, we re-use previous notation in the following formalisms.

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 denoted and 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*. Further, let* $\upsilon \in \mathbf{U}$ *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:*

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

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

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*

[12]We drop *implicit typing* [45] 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.[13] *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 $\big($`ohDoc:`$, \mathsf{get}($`ohDoc:`$)\big)$ 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.*, deref(`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 URIs 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, $\Gamma_1^R = \Gamma \cup \{(\upsilon, G_1^R)\}$, where υ 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, $\Gamma_2^R = \Gamma \cup \{(\upsilon, G_2^R)\}$, 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 $\Gamma \bullet R = \Gamma_3^R = \Gamma \cup (\upsilon, G_3^R)$. □

4. Link Traversal Based Query Execution

Having covered some necessary preliminaries, in the following section, we introduce the Link Traversal Based Query Execution (LTBQE) approach first proposed by Hartig *et al.* [29] for executing SPARQL queries over the Web of Data (§ 4.1). For this, we provide some formal definitions and examples (we tailor our definitions for the purpose of this work; a comprehensive study of the semantics and computability of LTBQE has been covered in [28]). We also highlight the assumptions under which the LTBQE approach works well.

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

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

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 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 the key notion of query-relevant documents in the context of LTBQE, and give an indication as to how these documents are derived. 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:* $\Gamma^Q_0 := \mathsf{derefs}(U_Q)$*. Thereafter, for* $i \in \mathbb{N}$*, define:*[14]

$$\Gamma^Q_{i+1} := \mathsf{derefs}\big(\mathsf{qrel}(Q,\Gamma^Q_i)\big) \cup \Gamma^Q_i$$

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

Example 5. We illustrate this core concept of LTBQE query-relevant sources with a simple example based on Figure 1. Let Q be the following query looking for the names of the authors of a named 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^Q_0. (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 $\mathsf{qrel}(Q,\Gamma^Q_0)$. Nothing else is matched. Hence, these two URIs are dereferenced and the results added to Γ^Q_0 to form Γ^Q_1.

LTBQE repeats the above process until no new sources are found. At the current stage, Γ^Q_1 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. (In)completeness of LTBQE

An open question is the *decidability* of collecting query-relevant sources: does it always terminate? This is dependent on whether one considers the Web of Data to be infinite or finite. For an infinite Web of Data, this process is indeed undecidable [28]. To illustrate this case, Hartig [28] uses the example of a Linked Data server describing all natural numbers[15], where each $n \in \mathbb{N}$ is given a dereferenceable URI, each n has a link to $n+1$ with the predicate `ex:next`, and a query with the pattern "`?n ex:next ?np1 .`") is given. In this case, the traversal of query-relevant sources will span the set of all natural numbers. However, if the (potential) Web of Data is finite, then LTBQE is decidable;

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

[15] Such a server has been made available by Vrandečíc *et al.* [65], but unfortunately stops just shy of a billion. See, *e.g.*, http://km.aifb.kit.edu/projects/numbers/web/n42.

in theory, it will terminate after processing all sources. The question of whether the Web (of Data) is infinite or not comes down to whether the set of URIs is infinite or not: though they may be infinite in theory [5] (individual URIs have no upper bound for length), they are finite in practice (machines can only process URIs up to some fixed length).[16]

Of course, this is a somewhat academic distinction. In practice, the Web of Data is sufficiently large that LTBQE may end up traversing an unfeasibly large number of documents before terminating. A simple worst case would be a query with a "open pattern" consisting of three variables.

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 first query-relevant sources will be identified as 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 Web of Data. Of course, this problem does not occur only for open patterns. One could also consider the following query which asks for the friends-of-friends of `oh:olaf`:

```
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.

Whether defined in an order-dependent or order-independent fashion, LTBQE will often not return complete answers with respect to the Web of Data [28]. We now enumerate some of the potential reasons LTBQE can miss answers.

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 ($\mathrm{deref}(\texttt{cb:chris}) = \emptyset$) and thus, the query processor cannot compute and select relevant sources from interim results.

Unconnected query-relevant documents: Similar to the previous case of reachability, the number of results might be affected if query relevant documents cannot be reached. This is the case if answers are "connected" by literals, blank-nodes or non-dereferenceable URIs. In such situations, the query engine cannot discover and dereference further query relevant data. 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"`.

[16]It is not clear if URIs are (theoretically) finite strings. If so, they are countable [28].

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 completeness (or, perhaps more fittingly, *recall*), Hartig *et al.* [29] proposed an extension of the set of query relevant sources to consider `rdfs:seeAlso` links, which sometimes overcomes the issue of URIs not being dereferenceable (as per `cb:chris` in our example). In the LiDaQ system, we include this extension, and on top, we propose further novel extensions that apply reasoning over query-relevant sources to squeeze additional answers from query-relevant sources, which in turn may lead to recursively finding additional query-relevant sources.

First, we propose following `owl:sameAs` links, which, in a Linked Data environment, are used to state that more information about the given resource can be found elsewhere under the target URI. Thus, to fully leverage `owl:sameAs` information, we first propose to follow relevant `owl:sameAs` links when gathering query-relevant sources and subsequently apply `owl:sameAs` reasoning, which supports the semantics of *replacement* for equality, meaning that information about equivalent resources is mapped to all available identifiers and made available for query answering.

Second, we propose some lightweight RDFS reasoning, which takes schema-level information from pertinent vocabularies and ontologies that describe the semantics of class and property terms used in the query-relevant data and uses it to infer new knowledge. In a first step, we must make schema data available to the query engine, where we propose three mechanisms:

1. a static collection of schema data are made available as input to the engine;
2. the properties and classes mentioned in the query-relevant sources are dereferenced to dynamically build a direct collection of schema data; and
3. the direct collection of dynamic schema data is expanded by recursively following links on a schema level.

Using the schema data collected by one of these methods, in the second step, we apply rule-based RDFS reasoning to materialise inferences and make them available for query-answering.

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:

First, let us motivate and give the intuition for the legacy `rdfs:seeAlso` extension with a simple example.

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

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

The query processor evaluates this query by dereferencing the content of the query URI `oh:olaf`, following and dereferencing all URI bindings for the variable `?f` and matching the second query pattern "`?f foaf:depiction ?n .`" over the retrieved content to find the pictures. However, the query processor needs to follow the `rdfs:seeAlso` link from `cb:chris` to `cbDoc:` since the URI `cb:chris` is not dereferenceable (recall that a dashed arrow in Figure 1 denotes dereferenceability). In summary, in this example, to find the document `cbDoc:` and ultimately answer the query, LTBQE needs to be extended to follow `rdfs:seeAlso` links. □

As such, Hartig *et al.* [29] proposed an extension of LTBQE to follow `rdfs:seeAlso` when looking for query-relevant sources. We briefly formalise this extension:

Definition 8 (LTBQE Extension 1: `rdfs:seeAlso`)**.** *Reusing notation from Definition 7, given a dataset Γ and a set of URIs U, first define:*

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

Now, for a given query, expand the set of query relevant sources as follows:

$$\mathsf{qrel}'(Q,\Gamma) := \mathsf{qrel}(Q,\Gamma) \cup \mathsf{seeAlso}\big(\Gamma, \mathsf{qrel}(Q,\Gamma)\big)$$

The rest of the definition follows from Definition 7 by replacing qrel(.) *with the extended function* qrel′(.).

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

We next formalise and describe our novel extension for following `owl:sameAs` links and applying equality reasoning. We again begin with a motivating example to cover the intuition.

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

```
SELECT ?f WHERE {
  oh:olaf foaf:knows ?f .
  ?pub dc:creator ?f , oh:olaf . }
```

Executing this query over the data in Figure 1 using the baseline LTBQE approach will not return any answers, since explicit equality information about URIs is not supported. The `owl:sameAs` relationship between `oh:olaf` and `dblpA:Olaf_Hartig` states that both URIs are equivalent and referring to the same real world entity, and hence that the information for one applies to the other. In summary, to answer this query, LTBQE must be extended to follow `owl:sameAs` links and apply reasoning to materialise inferences with respect to the semantics of replacement. □

We now formalise the details of this novel extension.

Definition 9 (LTBQE Extension 2: `owl:sameAs`)**.** *We propose an extension of LTBQE to consider `owl:sameAs` links and inferences. First, given a set of URIs U and a dataset Γ, as before, define:*

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

And define by extension:

$$\mathsf{qrel}''(Q,\Gamma) := \mathsf{qrel}(Q,\Gamma) \cup \mathsf{sameAs}(\Gamma, \mathsf{qrel}(Q,\Gamma))$$

By replacing qrel(.) *with* qrel′′(.)*, this latter function is used to find addition query-relevant sources by analogue to Definition 7.*

Now we must define the role of inference. Let R denote the set of rules of the form EQ-* *in Table 1. Extending the notation from Definition 7, define:*

$$'\Gamma_i^Q := \Gamma_i^Q \bullet R$$

In other words, $'\Gamma_i^Q$ denotes the closure of the raw Γ_i^Q dataset with respect to `owl:sameAs` data. The full `owl:sameAs` extension is then described by replacing qrel(.) *with* qrel′′(.) *(to follow `owl:sameAs` links) and Γ_i^Q with $'\Gamma_i^Q$ (for all i; to apply inferencing at each stage) in Definition 7.*

5.1.3. *Incorporating RDFS schemata and reasoning*

Finally, we formalise and describe our novel extension for retrieving and reasoning with respect to RDFS descriptions of classes and properties used in the query-relevant data. We again start with a motivating example.

Example 9. Take the following query, which asks 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 `"Olaf Hartig"`. 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 propose an extension of LTBQE to consider RDFS schema data and inferences. Let Ψ denote an auxiliary Linked Dataset that contains some schema data. Let R denote rules* {PRP-SPO1, PRP-DOM, PRP-RNG, CAX-SCO} *in Table 1 (other finite RDFS rules can be added as necessary). Extending the notation from Definition 7, define:*

$$''\Gamma_i^Q := \Gamma_i^Q \cup \Psi_i \bullet R$$

In other words, $''\Gamma_i^Q$ denotes the closure of the raw Γ_i^Q dataset and the schema data in Ψ_i at each stage (the subscript on Ψ_i indicates that schema data can be collected recursively, on the fly). The RDFS extension is then described by replacing Γ_i^Q with $''\Gamma_i^Q$ (for all i; to apply inferencing at each stage) in Definition 7.

We are then left to describe how Ψ_i may be acquired, where we provide three options (Ψ_i^{1-3}).

1. *A static corpus of schema data Ψ can be provided as input, such that $\Psi_i^1 := \Psi$.*
2. *The class and property terms used to describe $''\Gamma_i^Q$ can be dereferenced. Letting $\mathsf{preds}(\Gamma)$ denote the set of all URIs mentioned in the predicate position of some triple in the merge of Γ, and letting $\mathsf{o\text{-}type}(\Gamma)$ denote the set of all URIs mentioned in the object position of a triple in the merge of Γ whose predicate is `rdf:type`, we can define Ψ_i^2 as:*

$$\Psi_i^2 := \mathsf{derefs}\big(\mathsf{preds}(\Gamma_i^Q) \cup \mathsf{o\text{-}type}(\Gamma_i^Q)\big)$$

   or, in other words, the schema data obtained by directly dereferencing all predicates and values for `rdf:type` mentioned in the query-relevant data thus far.
3. *Finally, it is possible to extend the schema data by following schema-level links. For a Linked Dataset Γ, let $\mathsf{sl}(\Gamma)$ be a "schema links" function which extracts the set of all URIs u such that u is the subject or object of some triple $t \in \mathsf{merge}(\Gamma)$ with predicate `rdfs:subPropertyOf`, `rdfs:subClassOf`, `rdfs:domain` or `rdfs:range`; or u is the object of some triple $t \in \mathsf{merge}(\Gamma)$ with `owl:imports` as predicate. Now, taking Ψ_i^2 as above, let $\Psi_{i,0}^3 := \Psi_i^2$, and thereafter, for $j \in \mathbb{N}$ define:*

$$\Psi_{i,j+1}^3 := \mathsf{derefs}\big(\mathsf{sl}(\Psi_{i,j}^3)\big) \cup \Psi_{i,j}^3$$

   such that links are recursively followed up to a fixpoint: the least j such that $\Psi_{i,j}^3 = \Psi_{i,j+1}^3$. This fixpoint then represents Ψ_i^3. In other words, the third method of collecting schema extends upon the second method by recursively following core RDFS links and `owl:imports` links from the direct schema data.

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

In order to build the LiDaQ prototype, we have re-implemented Hartig *et al.*'s iterator-based algorithm for LTBQE [29] together with some optimisations such as a local per-query cache with an in-memory quad store [31] and a more efficient join operator [40]. We then add our various extensions. The code-base is written in Java. Our architecture—depicted in Figure 3—features five main components:

***Query Processor*:** uses the Java library ARQ to parse and process input SPARQL queries and format the output results.[17]

***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 delay between two consequential lookups for URIs hosted at the same domain to avoid hammering remote servers.[18]

***Local Repository*:** a custom implementation of an in-memory quad store is used to cache the content of all query relevant data, including inferences, as well as indexing triple patterns from the query to match against the data. Triple patterns are matched in a continuous fashion as new content is pushed to the cache, feeding the iterators. When reasoning is not enabled, the triple pattern index filters non-matching triples and discards them. This repository is also used to store static schema data, where needed.

***Reasoner*:** the Java-based SAOR reasoner is used to support the aforementioned rule-based reasoning extensions [11], and can execute inferences over the new content as it arrives in conjunction with the cache.

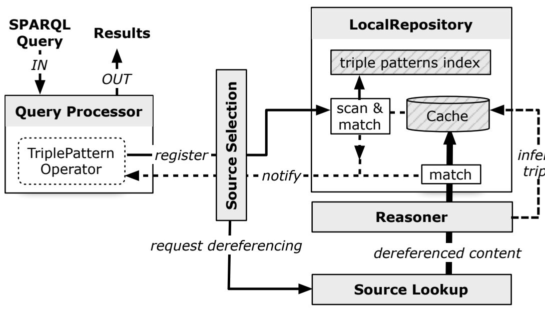

Fig. 3. LTBQE architecture diagram.

We further investigate a "reduced source-selection" variant of LiDaQ that uses straightforward, generic optimisations to minimise the number of sources considered while maximising results. Primarily, we avoid dereferencing URIs that do not appear in join positions: these are variables which are not used elsewhere in the query, but whose existence needs to be asserted. 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, this query will first visit `ohDoc:`, then bind `cb:chris` for `?f`, and then visit `cbDoc:` for more information about `cb:chris`. Here, `"Chris Bizer"` will be bound for `?fn`, and `dbpedia:Berlin` bound for `?b`. However, dereferencing the latter URI would be pointless: we do not need any information about `dbpedia:Berlin` to answer the query. Here the variable `?b` does not appear in other triple patterns and further information about URIs bound to it are not directly required by the query. Our optimisation proposes to avoid wasting lookups up not dereferencing URIs bound to non-join variables. □

Of course, by reducing the amount of sources and raw data that are accessed—and given that anyone in principle say anything, anywhere—we may also reduce the number of answers that are returned. Taking the previous example, for all we know, the document dereferenced through `dbpedia:Berlin` may contain (unrelated) information about friends of `oh:olaf`, which 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 posit that for real-world Linked Data, 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 Li-

[17] http://jena.sourceforge.net/ARQ/
[18] http://code.google.com/p/ldspider/

DaQ's source selection to prune remote lookups that are unlikely to contribute answers.

6. Empirical Study

We now begin to look at how LTBQE and its extensions can be expected to perform in practice. In Section 4.1, we mentioned that the recall of the LTBQE approach is—in the general case—dependent on the dereferenceability of data. Along these lines, we can draw general conclusions about the effectiveness of LTBQE for answering queries over the Web of Data by looking at the nature of dereferenceability on the Web of Data. In particular, we are interested to know the ratio of information available in dereferenceable documents versus the information available on the rest of the Web of Data. This gives us a indication 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 similarly test how much additional raw information is made available by our extensions.

In summary, we take a large crawl of the Web of Data as a sample. We survey the ratio of all triples mentioning a URI in our corpus against those returned in the dereferenceable document of that URI; we do so for different triple positions. We also look at the comparative amount of raw data about individual resources considering (1) explicit, dereferenceable information; (2) including `rdfs:seeAlso` links [29]; (3) including `owl:sameAs` links and inferences; (4) including RDFS inferences with respect to a static schema.

6.1. Empirical corpus

For our corpus, we take the dataset crawled for the Billion Triple Challenge 2011 (BTC'11) in mid-May 2011. The corpus consists of 7.4 million RDF/XML documents spanning 791 pay-level domains (data providers). URIs extracted from all RDF triples positions, but without 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 bulk of RDF data is serialised as N-Quads [14], which provides information regarding which triple came from which document in a manner directly analogous to our notion of a Linked Dataset.

Alongside the bulk of 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, with respect to the Web of Data, our high-level recall measures for LTBQE specify an *upper* bound: more information may be available on the Web of Data than we know about in our sample.

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 [11], schema data on the Web is often noisy, where third-party publishers "redefine" popular terms outside of their namespace; for example, one document defines nine *properties* as the domain of `rdf:type`, which would have a drastic effect on our reasoning.[19] Thus, we perform authoritative reasoning, which conservatively discards certain third-party schema axioms (cf. [11]). In effect, our schema data only includes triples of the following form:

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

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 data provided by 98 PLDs. We denote this dataset as $\Psi_\sim$.

[19] *viz.* http://www.eiao.net/rdf/1.0

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

6.3. Recall for Baseline

We first measure the average dereferenceability of information in our sample. Let $\mathsf{data}(u, G)$ return 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 mentioning d in some triple position. We define the *sample dereferencing recall* with respect to a sample graph 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 are interested to know how often relevant triples—*i.e.*, triples with that URI in the predicate position—occur in the dereferenced document, how many, and what ratio when 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`.

The analysis provides some interesting practical insights into the LTBQE approach. 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. In summary, LTBQE performs well when URIs appear in the subject position of triple patterns, moderately when URIs appear in the object on a non-type triple, but poorly when URIs appear in the predicate or object of a type triple.

One may also note the high standard-deviation values in Table 3: these indicate that dereferenceability is often "all or nothing", particularly for object and predicate URIs. In Figure 4, for all 745 dereferenceable predicate URIs, we plot the distribution of the number of triples in the dereferenced document that contain the dereferenced term in the predicate position (log/log scale). Figure 5 shows the analogous distribution for dereferenceable object URIs in type triples. 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 see that a few predicates and values for `rdf:type` return a great many relevant triples in their dereferenced documents.[20] This explains the high standard deviations: for example, although most predicates return no relevant information in their dereferenced document—and thus the probability of retrieving relevant information by dereferencing the predi-

[20]Many such examples for both classes and predicates come from the SUMO ontology: see, *e.g.*, `http://www.ontologyportal.org/SUMO.owl#subsumingRelation`

Table 3
Dereferenceability results for different triple positions

Position	$\|U_\sim\|$	$\|D_\sim\|$	$\frac{\|D_\sim\|}{\|U_\sim\|}$	sdr$_\sim$		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

cate URI in a triple pattern is low—a few predicates dereference to large sets of relevant information.

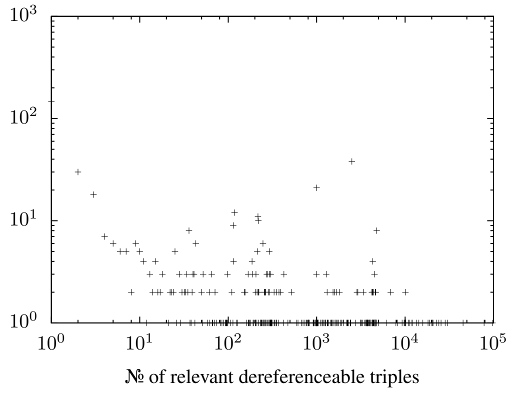

Fig. 4. Relevant dereferenceable triple distribution for predicate URIs (log/log)

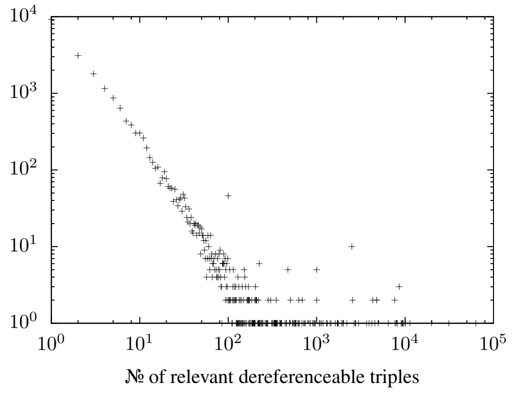

Fig. 5. Relevant dereferenceable triple distribution for type-object URIs (log/log)

6.4. *Recall for Extensions*

We now look at how the three LTBQE extensions can help to find additional data for generating query answers. Table 4 presents the average increase in raw information 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 $D_\sim$ which 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, ddata$^+_\sim$ indicates the analogous ddata$_\sim$ measure after the extension has been applied: this may involve data outside of the dereferenced document, such as documents reached through `rdfs:seeAlso` or `owl:sameAs` links, or static schema data.

6.4.1. *Benefit of following* `rdfs:seeAlso` *links*

We measured the percentage of dereferenceable URIs in $D_\sim$ which have at least one `rdfs:seeAlso` link in their dereferenced document to be $\sim 2\%$ for our sample (about 201 thousand URIs). 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, in the general case, considering `rdfs:seeAlso` information for the query processing will only marginally affect the recall increase of LTBQE.

6.4.2. *Benefit of following* `owl:sameAs` *links & including inferences*

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 6

Table 4
Additional raw data made avaiable 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
`rdfs:seeAlso` LINKS	2.01×10^5	0.02	1.006	± 0.04
`owl:sameAs` LINKS & INFERENCE	1.35×10^6	0.16	2.5	± 36.23
RDFS INFERENCE	6.79×10^6	0.84	1.8	± 0.76

(log/log), which shows the distribution of the ratio of increase by considering `owl:sameAs` for individual URIs: we again see that although the plurality of URIs enjoy a small increase in raw data, a few URIs enjoy a very large increase. In more detail, Figure 7 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.

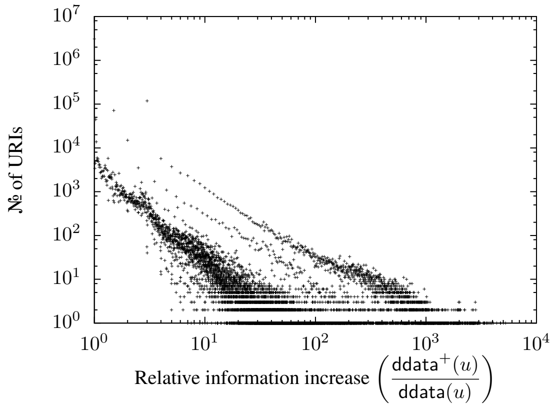

Fig. 6. Distribution of relative information increases by materialising `owl:sameAs` information (log/log)

We conclude that, in the general case, `owl:sameAs` links are not so commonly found 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 including RDFS reasoning*

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

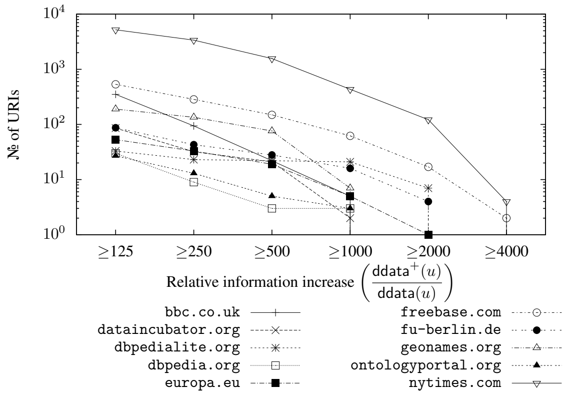

Fig. 7. Binning of relative information increases by materialising `owl:sameAs` information per domain (log/log)

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*

Without 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 use these results to modify our variant of LiDaQ which tries to minimise wasted remote lookups: aside from skipping URIs bound to non-join variables, 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). Of course, we still dereference these URIs for the purpose of dynamically collecting a set of schemata when performing RDFS reasoning.

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 we 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

We now look at existing SPARQL benchmark frameworks and how they have been used to evaluate various "live" Linked Data query processing approaches in the literature. The main purpose of this survey is to study the query types and the benchmark environments used in order to inform our own evaluation of LTBQE and its extensions. We find that various published SPARQL benchmark frameworks focus on Linked Data query processing and some of the provided queries could be re-used, but that existing papers evaluate their approaches with respect to either hand crafted queries or domain-specific queries, whereby all are restricted to one or few domains of data.

7.1.1. Benchmark Frameworks

We now discuss existing Linked Data SPARQL benchmarking frameworks. Since we aim to run our evaluation over Linked Data sources *in situ*—and to demonstrate the real world behaviour of the methods discussed herein—we focus on query frameworks designed to run over real-world Linked Data, where we do not treat SPARQL benchmarks designed to run over synthetic datasets such as *LUBM* [22], *BSBM* [10], or *SP^{2}Bench* [52, 53]. In fact, to the best of our knowledge, only two relevant frameworks have been proposed:

FedBench The FedBench [51] framework is designed specifically for testing Linked Data querying scenarios. Queries are formulated against three data collections [51, Table 1]:

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 [52, 53]; and
3. a general Linked Open Data Collection, which includes datasets like DBpedia, GeoNames, Jamendo, LinkedMDB, The New York Times and Semantic Web Dog Food.

Three independent query sets are then defined. 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 (e.g join complexities), and so forth. The second query set consists of the original SP^{2}Bench queries. The third query set provides 11 Linked Data specific queries[21], which consist of SPARQL Basic Graph Patterns in the form of 6 path queries, 3 star queries and 2 mixed/hybrid queries. This third set is thus of particular interest to us.

7.1.2. Evaluation of Linked Data Query Approaches

We now look at the specifics of how various authors have evaluated their proposed approaches for querying Linked Data. We focus on the evaluation of non-materialised engines: *i.e.*, we focus on query proposals that (typically) involve accessing remote data at runtime. In particular, we summarise which query sets were used, what data were used, and how the benchmark was setup.

LTBQE1 In the work which initially proposed the explorative LTBQE model, Hartig *et al.* [29] used four manually crafted queries to demonstrate the feasibility of the approach in the real-world. The results present query time, number of results and number of sources involved. In addition, the au-

[21] http://code.google.com/p/fbench/wiki/Queries#Linked_Data_(LD)

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

Table 5
Summary about number of queries, evaluation measures and setup in the literature about live Linked Data query methods.

thors used 12 BSBM queries (for synthetic data) and a controlled setup with a proxy server to evaluate the query time for different fetch-scheduling strategies.

LTBQE2 This work studies how different in-memory data structures might influence the performance of the original LTBQE approach [31]. The proposed data structures are evaluated with different datasets and the query performance was again benchmarked with BSBM.

LTBQE3 Another proposed LTBQE extension is to cache query relevant data to improve the result completeness of the LTBQE approach [26]. The published evaluation uses a real world use-case, called the FOAF Letter application[22], which involved five query templates—with a mix of different shapes and SPARQL features—instantiated for 23 people, giving a total of 115 queries[23].

LTBQE4 In follow up work, Hartig [27] discusses and evaluates different strategies for the query execution order. The impact of the query evaluation order is evaluated by executing 3 manually crafted real-world queries six times live over the Web of Data. The three queries are a mix of star and path-shaped Basic Graph Patterns with 6–8 elements.

[22]The application provides a service for users with FOAF profiles to keep track of their social network by periodically checking updates and suggesting new connections using the LTBQE approach. See `http://linkeddata.informatik.hu-berlin.de/foafletter/`

[23]`http://squin.sourceforge.net/experiments/CachingLDOW2011/`

LT10 Ladwig and Tran [40] investigate three different strategies to execute SPARQL queries over the Web of Data and systematically analyse and compare them. The benchmark contains 8 queries and was executed in a controlled environment with a local proxy server.

SIHJoin Ladwig and Tran [40] present experiments on real-world datasets and on synthetic data to evaluate the benefit of their proposed symmetric hash-join operator against the non-blocking iterator proposed by Hartig *et al.* [29]. Their evaluation is executed in a controlled environment (hosted using CumulusRDF [38]) and uses 10 manually created queries (similar to the FedBench Linked Data query set).

FedX Schwarte *et al.* [55] evaluate their proposed system for federating SPARQL endpoints using the (entire) FedBench SPARQL benchmark framework.

LH10 Li et.al. [43] investigated using reformulation trees to organise local summarised knowledge about sources; they also investigate how OWL reasoning can be used for the explorative query execution. The evaluation uses a synthetic data set and manually selected subsets of the Billion Triple Challenge 2010 dataset. Queries are created manually. The experiment was executed in a controlled environment and HTTP lookups were simulated.

SPLENDID Görlitz and Staab [21] evaluated their SPARQL federation approach (based on VoID descriptions) using the life science and cross domain query sets from FedBench. The benchmark was conducted over a local replication of the datasets and endpoints.

SPARQL-DQP Buil Aranda *et al.* [3] evaluate their federated SPARQL techniques using modified version of the Life Science queries in the FedBench framework, where additional SPARQL features are added. The queries are then run over four endpoints, two of which are replicated locally.

QTree In previous works [61], we compare several data structures and hash functions with respect to their suitability for determining relevant sources during query evaluation. We experimented with queries that are automatically generated from random walks over real-world data, corresponding to “star-shaped" and “path-shaped" queries (similar to QWalk presented later). We evaluated the system in an simulated and controlled environment.

To provide a good overview—and for ease of comparison—we summarise this survey of the evaluation of Linked Data query systems in Table 5.

7.1.3. Discussion

We see that there is a lot of diversity in how different Linked Data query proposals have been evaluated. First and foremost, we highlight that few benchmarks have been proposed for real-world Linked Data; perhaps the most agreed upon is FedBench. Furthermore, most benchmarks and evaluations involve either a handful of manually crafted queries designed to run over a small number of sources (FedBench), or involve a larger number of (semi-)automatically generated queries but are tied to a specific domain (*e.g.*, DBPSB) or schema of data (*e.g.*, FOAF Letter). Furthermore, we note that few engines run their queries live over remote resources, but rather prefer to replicate raw content or endpoints within a controlled environment. In summary, we find that 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], which was run live, but which involved a handful of resources and was centred specifically around FOAF profiles.

7.2. QWalk: Random Walk Query Generation

Given the shortcomings of existing benchmarks, we propose a new benchmark framework—called *QWalk*—that is tailored to link-traversal based approaches and builds a large set of queries that are answerable over a diverse set of real-world sources that use different schemata. 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, such that it should return results through an LTBQE-style approach.

7.2.1. Query shapes

To inform the types of queries we generate, we take observations from the work of Gallego *et al.* [19], 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.

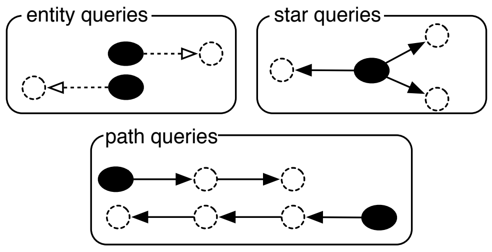

Fig. 8. Visualisation of example benchmark queries (entity-s, entity-o, s-path-2, o-path-3, star-2-1); dotted lines represent query variables

Along similar lines, for our benchmark we generate queries of elemental graph shapes as depicted in Figure 8, *viz.,* entity, star and path queries. We now describe these query types in more detail.

Entity queries (entity-<s|o|so>) ask for all available triples for an entity. We generate three types of entity queries, asking for triples where a URI appears as the subject (entity-s); as the object (entity-o); as the subject *and* object (entity-so). These type of queries are very common in Linked Data Browsers, interfaces, or for dynamically serving dereferenceable Linked Data content. An example for entity-so would be

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

Star queries (star-<s3|o3|s1-o1|s2-o1|s1-o2>) have three acyclic triple patterns which share exactly one URI (called the centre node) and where predicate terms are constant. We do not consider stars involving predicates since (as discussed) they are rarely used in practice [19]. We generate four different variations of such queries, differing in the number of triple patterns in which the centre node appears at the subject (s) or object (o). Thus, each query has 4 constants and 3 variables. 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 which form a path such that precisely two triple pattern share the same variable. Exactly one triple pattern has a URI at either the subject or object position and all predicate terms are constant. As before, we do not consider paths connected through predicate variables since they are rare [19]. 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 pattern is a constant. An example for s-path-2 is the following:

```
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 we have seen, most works have evaluated LTBQE-style approaches in controlled environments—using proxies and simulated remote access—or only for a small number of queries or sources in uncontrolled environments. Conversely, we now wish to stress-test LTBQE—and our proposed reasoning extensions—for a large range of diverse queries in uncontrolled environments, and to characterise how these methods handle the aforementioned challenges.

Based on the discussion of the previous section, we select three complementary benchmarks to run with LiDaQ:

1. FedBench Linked Data Queries, which offers a few manually crafted queries designed to run over a number of different domains;
2. DBpedia SPARQL Benchmark (DBPSB), which offers the potential to generate many queries designed to run over one domain, and are based on real-world query logs;
3. QWalk, which offers a large selection of queries that can be run directly over a diverse set of sources.

We execute all queries directly over the Linked Data Web in an uncontrolled environment without any replication, proxies or simulation of HTTP lookups. Thus, the measured values reflect the expected query behaviour in a real application scenario and allows us to discuss the feasibility of LiDaQ—and of LTBQE query approaches in general—within such a setting.

We first discuss the measures that we take and the environment in which the experiments are run (§ 8.1). We then introduce the configurations of LiDaQ that we evaluate (§ 8.2). Then we present the results of the FedBench run (§ 8.3), the DBPSB run (§ 8.4), and the QWalk run (§ 8.5).

8.1. Environment and Measures Taken

All evaluation is run on one server with 4 GB of main memory. 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.

Given that we run queries live over a diverse set of remote sources, we often encounter some "non-deterministic" behaviour: in particular, a source may be readily accessible during some query runs, but may be unresponsive or not return at all in others. In other words, 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*, which we consider in our results 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 following measures:

result: the number of distinct results,
time: the total time taken to execute the query,
first: time elapsed until the first result was returned,
lookups: total number of HTTP GET lookups,
data: total number of raw triples retrieved, and
inferred: total number of unique triples inferred.

These measures together characterise the behaviour of the LiDaQ engine while processing queries under various setups. However, raw counts of query results can sometimes exhibit outliers due to products inherently caused by joins. We illustrate this with an example QWalk query, which caused some surprising behaviour when RDFS reasoning was used.

Example 11. For QWalk, we encountered the following 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 $1 \times 201 \times 199 = 39,999$ results, giving a two orders of magnitude increase. □

The query results for the above example involves a lot of repetitions of terms: each term bound to `?s0` or `?s1` would appear about 200 times. Hence we add an additional measure to help loosely characterise the "redundancy-free content" of the results: we summate 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 $1 + 201 + 199 = 39,999$ 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`. We expect in theory the smallest amount of results and also the fastest query time. The following extensions are built upon this reduced profile, 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 the URIs contain these information in the dereferenceable document and that they make little additional raw data available.

With `owl:sameAs` links and inference (SAMEAS)**:** this benchmark setup extends the CORE$^-$ setup by considering `owl:sameAs` links and inference. We expect for all queries 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 information is 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 ease of reference, Table 6 gives an overview of 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. FedBench results

Our first experiment uses the FedBench Linked Data Queries (§ 7.1.1) 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 In initial tests, we only received results for 6 out of the 11 FedBench queries. A manual inspection of the queries and their relevant data revealed that the empty results were either caused by (i) access forbidden by `robots.txt` or (ii) updates in the DBpedia datasets (which, in itself, lends strength to the arguments for live querying approaches).

In the first category, there was only 1 query:

Disallowed by `robots.txt` (LD7): This query requires data from the `geonames.org` domain, especially from the subdomain `sws.geoname.org`. The query fails because the access for resources on this subdomain is disallowed via the `robots.txt` protocol.[24]

We do not wish to contravene the Robots Exclusion Protocol and so we do not run this query in the main evaluation.

In the second category—those caused by being out-of-sync with DBpedia—there were 4 queries, which we fixed manually:

Missing language tag (LD9): One query was missing a language tag. We changed the query literal `"Luiz Felipe Scolari"` by adding the English language tag: `"Luiz Felipe Scolari"@en`.

Outdated predicates (LD8–10): Three queries contain the predicate `skos:subject`, which was initially proposed for SKOS and used by DBpedia. However, the SKOS Working Group later chose to instead reuse `dcterms:subject` and DBpedia followed suit. We thus changed the query predicates from `skos:subject` to `dcterms:subject` where applicable.

Thus, we keep 10 of the 11 original queries, where 4 are adapted slightly to work against current versions of DBpedia, and where 1 is dropped due to `robots.txt` issues. In addition, since we count results, we add to all queries the DISTINCT solution modifier to eliminate duplicates. Our updated queries are online[25]. However, we still find that some of the updated queries do not return results: queries LD6 and LD9 return 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).

Overview of Experiments In total, using LiDaQ, for each configuration, 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). 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.[26] Herein, we focus on the general impact of our extensions on the repeatability and reliability of the results across the four runs, averaged across all queries.

Detailed Results Given the number of queries (10), configurations (11) and measures (6), we rather present detailed discussion of the FedBench results for each individual query in Appendix A.

Summarising herein, 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, we find that 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

[24]See http://sws.geonames.org/robots.txt, which states that all agents are disallowed.

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

[26]We found queries for which it sends at least eight lookups per second to the same server.

LD11) and results require a product for variables with compatible mappings [47]. 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; as discussed previously, 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 In summary, 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.[27] For DBpedia, the schema descriptions of class and property terms are hosted in individual documents and often interlink with related sources like Yago [58] and CYC, leading to many documents being requested when schemata are dynamically retrieved, leading to unstable behaviour for such configurations with respect to these queries. Given that the queries are restricted to a few data providers, politeness policies play a crucial role: the amount of time-delay enforced between subsequent lookups to the same host can be a major factor for performance.

In general, 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 involve 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.

[27]We observed and reported such problems before: see `https://groups.google.com/forum/?fromgroups#!topic/pedantic-web/rXQPcFLMOi0` for detailed discussion.

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

8.4. *DBPSB Results*

The FedBench Linked Data queries are designed specifically for evaluation using LTBQE-style engines such as LiDaQ. We now rather look at the DBpedia SPARQL benchmark [44] (DBPSB; § 7.1.1): a generic Linked Data SPARQL benchmark containing realistic queries based on popular patterns mined from real-world DBpedia access logs. Thus, DBPSB should give us an indication as to how well LTBQE can cope with (non-tailored) queries that are based on those frequently run by users against the materialised DBpedia SPARQL engine. As we will see, LTBQE and its extensions stuggle for this query suite.

Query Testing In total, the DBPSB query set consist of 25 different query templates (denoted DB1–25).[28] Each template has an associated template query that can be run against a DBpedia SPARQL endpoint to instantiate a set of concrete queries used for evaluation: each template query has a subset of *template variables* that are bound to create a new evaluation query (one query per binding) in this manner, where the rest of the variables are left as is for the evaluation query.

In a first step, we manually inspected the DBPSB query templates and ruled out those which LTBQE would clearly not be able to answer. We thus initially ruled out 16 queries:

Unsupported `OPTIONAL` feature (8 queries)**:** A total of 8 query templates use the `OPTIONAL` keyword, which can only be correctly evaluated over a closed dataset since it is a non-monotonic feature [28]. Problematically, if data are not available to match the `OPTIONAL` clause, SPARQL specifies that `UNBOUND` should be returned. Returning `UNBOUND` is a definitive answer that the data are not available, and can be tested elsewhere in a `FILTER` clause, allowing features such as negation-

[28]Originally taken from http://dbpedia.aksw.org/benchmark.dbpedia.org/Queries2011.txt. Formatted and annotated templates can be found at http://code.google.com/p/lidaq/source/browse/queries/dbpsb.txt

as-failure. LTBQE cannot definitively say that data are not available (unless a bounded dataset is considered [28]).

No suitably deref. URI (8 queries)**:** As discussed previously, dereferencing class and predicate URIs rarely returns their extension, and this is the case for DBpedia; for example, looking up the class `dbo:SoccerPlayer` does not return all instances of soccer-player, and looking up the predicate `dbo:thumbnail` does not return all relations between things and their thumbnails. A total of 8 query template instances would *only* involve URIs in such positions. (We do not include a further 4 `OPTIONAL` queries that also do not contain a suitably dereferenceable URI.)

This is perhaps an interesting observation in itself: only 9 of the 25 DBPSB query templates (36%) mined from real-world query logs could potentially be answered by LTBQE. If we were to relax the restriction on `OPTIONAL` and run it in a "best-effort" manner—or with a closed dataset semantics—LTBQE could run potentially run 13 of the 25 DBPSB queries (52%).

Overview of Experiments For the evaluation, we wanted to generate 25 sample queries for each of the remaining 9 DBPSB templates. For this, we ran the template queries provided for the benchmark against the public DBpedia SPARQL endpoint[29] and generated up to 1,000 results. From the template results, we randomly selected 25 to generate the query instances. Of the 9 templates, we encountered problems instantiating another 3 due to problems with DBPSB and the DBpedia endpoint itself:

Could not generate 25 instances (4 queries)**:** We did not get 25 instances for 4 of the templates using the public SPARQL endpoint. Of these, 2 template queries repeatedly timed out and thus could not be instantiated. The other 2 query templates returned insufficient results to generate enough concrete queries: both queries generated only two instances due to the use of the predicate `dbpprop:redirect`, which returns only two triples from the public endpoint.

One of the queries that could not be instantiated involved `UNION` patterns such that it could be run and still generate results without the template query variable being instantiated, so we include this in the evaluation (DB17). As such, we are left with only 6 DBPSB templates that are usable for evaluating our methods.[30] We had problems with another template query: the query plan for DB24 was ordered in such a way that the only dereferenceable URI was in a class position. Hence we are only left with 5 runnable templates.

We benchmarked 6 of the 10 LiDaQ setups: based on experiences with unstable DBpedia queries in FedBench and some initial runs for DBPSB, we dropped configurations involving the dynamic collection of schema data as they increased the demand of sources from DBpedia and exacerbate unstable behaviour (*cf.* Table 7). We thus focus on evaluating CORE, CORE$^-$, SEEALSO, SAMEAS, RDFS$_s$ and COMB$_s$. Queries are run live and directly over dereferenceable DBpedia data *in situ*; some queries may also traverse links from DBpedia and find additional answers remotely.

Table 8 shows high-level statistics about the results retrieved for the query instances generated for each template. We show the number of queries which returned some results, divided by those considered to be benchmark stable (see § 8.1) and those which were not; and we also show the number of queries for each template which did not return results. The templates that generated queries with empty results involved `UNION` patterns with shared template queries, which caused problems that we discuss later, particularly for DB4.

Table 8
Statistics about stability per DBPSB query template

Template	**Total**	**Non-Empty**		**Empty**
		stable	*unstable*	
DB1	25	23	2	0
DB4	25	0	12	13
DB5	25	0	13	12
DB13	25	24	0	1
DB17	25	24	1	0

Detailed Results The results of the DBPSB experiments are given in Table 9, with average measures given across all query instances. For each query template class, we now discuss the results. Herein, variables marked like "`%%var%%`" are template variables, which are instantiated to create concrete instances of queries.

[29]http://dbpedia.org/sparql/

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

DB1: RETURN THE TYPE(S) OF A CERTAIN ENTITY

```
SELECT DISTINCT ?var1
WHERE {
  %%var%% rdf:type ?var1 .
}
```

This simple query consists of only one triple pattern with a URI in the subject and predicate position (DBpedia does not contain blank nodes in the template variable position). LiDaQ results for instances of this template are listed in Table 9. We see that each query returns on average about 3 results per query without reasoning. We see that our reduced source selection optimisation works well (CORE$^-$ and the extensions based on it), reducing the average number of HTTP GET requests from 8 to 2 (a single source lookup including redirect) and thus requiring only $\sim$20% of the time taken for CORE (the additional lookups are for the predicate `rdf:type` and for the bound class URIs). Furthermore, we see that reasoning also increases the number of results at the cost of additional query time: SAMEAS sees only a minor increase in results, but RDFS$_s$ and COMB$_s$ more than double the results by inferring additional types through sub-class, domain and range semantics. The extensions supporting `owl:sameAs` require looking up (on average) approximately one additional source.

DB4: LIST THE NAMES OF ENTITIES WHICH ARE CONNECTED (IN EITHER DIRECTION) TO THE QUERY ENTITY.

```
SELECT ?var5 ?var6 ?var9 ?var8 ?var4
WHERE {
  { %%var%% ?var5 ?var6 .
    ?var6 foaf:name ?var8 .
  }
  UNION
  { ?var9 ?var5 %%var%% ;
       foaf:name ?var4 .
  }
}
```

Some of the generated queries failed to return results since they bind literals to the template variable due to the second mention of `%%var%%` in the object position. In such cases, these literals cannot be dereferenced and LTBQE cannot find results.[31] Table 9 shows the average results for all queries. Though SAMEAS generates some additional results, it also instigates some unstable behaviour (*cf.* Table 8), where we see a large number of triples being retrieved and inferred, and where we see that the number of lookups triples. We can also see the benefit of CORE$^-$ in reducing the number of lookups vs. CORE while not affecting results.

DB5: LIST THE NAME AND COMMENTS OF A GIVEN SERIES WITH A GIVEN TYPE; OR LIST THE NAME AND COMMENTS OF A SERIES WITH A GIVEN TYPE THAT REDIRECTS TO A GIVEN URL

```
SELECT DISTINCT ?var3 ?var4 ?var5
WHERE {
  { ?var3 dbpp:series %%var1%% ;
         foaf:name ?var4 ;
         rdfs:comment ?var5 ;
         rdf:type %%var0%% .
  } UNION {
    ?var3 dbpp:series ?var8 .
    ?var8 dbpp:redirect %%var1%% .
    ?var3 foaf:name ?var4 ;
         rdfs:comment ?var5 ;
         rdf:type %%var0%% .
  }
}
```

From Table 8, we encountered some similar behaviour as for the previous query: some bindings for the template variable `%%var1%%` were again literals, leading to query instances for which no results could be found through LTBQE. From the detailed results in Table 9, although SAMEAS and COMB$_s$ found additional results, they did so at the cost of causing unstable behaviour, increasing the number of HTTP lookups by a factor of $\sim 10\times$. Again we see the benefit of CORE$^-$ in reducing the number of lookups vs. CORE while not affecting results.

DB13: LIST ENGLISH COMMENTS, DEPICTIONS AND HOMEPAGES FOR AN ENTITY.

```
SELECT *
WHERE {
  { %%var%% rdfs:comment ?var0 .
    FILTER (lang(?var0) = "en")
  }
  UNION
  { %%var%% foaf:depiction ?var1 }
  UNION
  { %%var%% foaf:homepage ?var2 }
}
```

Much like DB1, this star-shaped query is quite straightforward for the LTBQE approach as the results in Table 9 illustrate. Again the reduced source selection (CORE) shows benefits, returning all results, but re-

[31]SPARQL does allow literals in the subject position, though not allowed by RDF.

ducing the amount of HTTP lookups: only one source needs to be retrieved (requiring two lookups including the redirect), and results take 2.67 seconds to process in this setup.

In this case, RDFS reasoning alone has no effect on the results, but increases the time by a factor of 3.4×. Support for `owl:sameAs` statements increases the result size by a factor of 6×, but at the cost of a 9.5× increase in time as an average of 5.5 additional sources are fetched. When `owl:sameAs` and RDFS support are combined in COMB$_s$, a few additional answers are found over SAMEAS alone.

DB17: LIST THE FRENCH LABELS FOR ENTITIES WITH THE SUBJECT EITHER GERMAN STATE CAPITALS, PREFECTURES IN FRANCE OR THE QUERY DEFINED SUBJECT.

```
SELECT DISTINCT ?var2 ?var3
WHERE{
  { ?var2 dcterms:subject %%var%%. }
  UNION
  { ?var2 dcterms:subject
              dbpcat:Prefectures_in_France . }
  UNION
  { ?var2 dcterms:subject
              dbpcat:German_state_capitals . }
  ?var2 rdfs:label ?var3.
  FILTER (lang(?var3)="fr")
}
```

We updated this query to use `dcterms:subject` instead of `skos:subject`.[32] However, the updated query template times out; hence we run the query without the first union clause containing the template variable. The results in Table 9 show that this query is more expensive to execute than star-shape queries with specific subject URIs. In particular, there are 99 prefectures listed for France and 15 German capital states, as well as the members of the category given by the template variable to dereference. Given the large number of documents accessed, we found that extensions following `owl:sameAs` links took too long to run for our experiments; hence these results are omitted.

Conclusion First, we notice that many of the DBPSB queries are unsuitable for LTBQE, and that we ended up only being able to run a small fraction of the original queries. Second, we generally found that CORE$^-$ offers good performance with respect to CORE, with minimal effect on results. Third, we found that RDFS$_s$ only had a significant effect for the first query, asking for the types of a given entity. Fourth, we found that following `owl:sameAs` links on DBpedia invoked high overhead and unstable behaviour for 3 of the 5 queries, but also found various additional answers (though primarily aliases of result URIs). Ultimately, we conclude that LTBQE and its extensions (particularly those involving reasoning) struggle to cope with the complexity of DBPSB queries, which are designed to put materialised engines through their paces.

[32]Although there are puzzlingly some `skos:subject` predicates in DBpedia, they are not used to relate entities to categories: `dcterms:subject` is now used in this case.

8.5. QWalk results

Having looked at the FedBench evaluation containing a few manually-crafted queries answerable by LTBQE over a small number of real-world sources, and the DBPSB queries based on real-world query logs answerable (mostly) over DBpedia, we now look at the QWalk benchmark (§ 7.2), which automatically builds a large set of queries answerable over a wide range of real-world sources. For this, 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 a total of 1,100 initial queries. As before, we then ran these live over remote sources in an uncontrolled setting using various configurations of LiDaQ.

Query testing We first wished to filter out queries that did not return any answers or that did not show benchmark stable behaviour.

To begin, for the entity 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 10. Though the stability of entity-o and entity-so queries are not significantly affected, the number of stable queries for entity-s queries more than halves. Furthermore, as we will see later, the dynamic import of schemata often requires over 10× the runtime of CORE, and over 5× the runtime of static schema equivalents. Due to problems with instability and long runtimes, and given the number of queries in the benchmark, we do not run the dynamic schema configurations for QWalk queries.

Table 9

Results for DBSPB queries DB1, DB4, DB5, DB13 and DB17

	Setup	Term		Results		Time (s)		First (s)	HTTP	Data	Inferred
		avg.	*σ*	*avg.*	*σ*	*avg.*	*σ*				
DB1	CORE	3.09	±4.66	3.09	±4.66	7.2	±9.4	2.4	8	3,034.74	—
	CORE$^-$	3.09	±4.66	3.09	±4.66	1.5	±0.7	1	2	6.3	—
	SEEALSO	3.09	±4.66	3.09	±4.66	1.6	±1	1.1	2	6.3	—
	SAMEAS	3.26	±5.06	3.26	±5.06	5.6	±8.4	1.5	3.13	203.87	143.83
	RDFS$_s$	6.57	±5.85	6.57	±5.85	10.8	±21.9	5.6	2	100.39	53.35
	COMB$_s$	6.87	±6.54	6.87	±6.54	11.8	±27.3	2.1	3.13	297.26	237.39
DB4	CORE	51	±73.56	24.75	±36.96	71.2	±103.5	71	147.5	5,427.75	—
	CORE$^-$	51	±73.56	24.75	±36.96	71	±103.1	70.7	139	2,739.5	—
	SEEALSO	51	±73.56	24.75	±36.96	71.6	±104.5	71.3	140.25	2,777.75	—
	SAMEAS	164.75	±209.84	174.5	±226.3	246.8	±327.7	246.4	801.5	104,864.25	94,473.25
	RDFS$_s$	51.75	±73.72	46.25	±72.74	108.9	±103.3	73.3	141	8,940	5,663.5
	COMB$_s$	165.5	±210.04	323.5	±444.02	291.8	±332.8	247.1	807.25	130,207.25	119,221.75
DB5	CORE	12.1	±13.74	7.7	±10.1	9.5	±4.3	5	21.7	689.4	—
	CORE$^-$	12.1	±13.74	7.7	±10.1	7.4	±4.2	3.8	13.7	187.4	—
	SEEALSO	12.1	±13.74	7.7	±10.1	7.5	±4.2	3.7	13.9	188.3	—
	SAMEAS	39.6	±41.8	993	±2,410.14	196.9	±381.3	24.6	133.4	104,446.5	113,258.6
	RDFS$_s$	12.1	±13.74	7.7	±10.1	45.4	±4.3	6.4	13.7	1,352.8	822.4
	COMB$_s$	39.6	±41.8	993	±2,410.14	200	±238.3	16	119.1	99,241.5	97,387.9
DB13	CORE	3	±0	2.67	±0.48	4	±2.5	1.9	8.08	261.42	—
	CORE$^-$	3	±0	2.67	±0.48	2	±2	1.6	2	13.46	—
	SEEALSO	3	±0	2.67	±0.48	2.5	±2.3	1.6	2	13.46	—
	SAMEAS	3	±0	15.96	±13.35	18.6	±3.6	2.1	7.5	1,487.12	1,140
	RDFS$_s$	3	±0	2.67	±0.48	6.7	±21.2	1.9	2	540.17	252.21
	COMB$_s$	3	±0	17.21	±13.49	23.4	±25.8	2.2	7.5	1,889.21	1,540.88
DB17	CORE	228	±0	114	±0	124.5	±0.5	121.7	237.92	4,679.75	—
	CORE$^-$	228	±0	114	±0	122.9	±3.5	5.4	234	4,420.33	—
	SEEALSO	228	±0	114	±0	143.3	±3	5.1	237.33	4,421	—
	RDFS$_s$	228	±0	114	±0	161.6	±4.3	5.1	234	125,408.25	68,798.75

Table 10

Stable entity queries with and without dynamic schema extensions

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

Thus, considering only CORE$^-$, CORE, SEEALSO, SAMEAS, RDFS$_s$ and COMB$_s$ configurations, and for each query shape, Table 11 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 that there are at least two URIs with 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.

Detailed Results We now look at the average measures for results across all (non-empty stable) queries per query class: we begin with entity queries, then progress to star queries and eventually to path queries. Detailed results for each of our measures can be found for reference in Tables 23–25 of Appendix B. Herein,

Table 11
Summary of stable/unstable/empty queries for QWalk benchmark

Class	**Non-Empty**		**Empty**					
	s.	*uns.*	*all*	*4XX*	*5XX*	*6XX*	*mix*	*data*
entity-o	57	7	36	18	2	11	0	5
entity-s	60	5	35	18	1	11	0	5
entity-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

we plot the total time 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). Again, absolute measures can be found in Tables 23–25. In general, we found a lot of variance and outliers in our results; hence we summarise results with bar plots which show the 50^{th}, 75^{th}, 90^{th} and 100^{th} percentiles of the result-sizes and times across the query classes, where the percentiles characterise how the majority of queries behaved, and what outliers occurred.

entity-*: GET GENERIC INFORMATION ABOUT A GIVEN RESOURCE

Entity 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 9 presents the increase in time over CORE$^-$ for all other configurations across the three classes of entity queries, broken down by percentiles, with the x-axis presented in log-scale, where the 10^0 line indicates no change from CORE$^-$. Figure 10 analogously presents the increase in query results returned versus CORE$^-$.

We can see from the 50^{th} percentile in Figure 9 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 entity query classes, with the most severe case (on the 100^{th} percentile) taking almost eight times longer for entity-s.

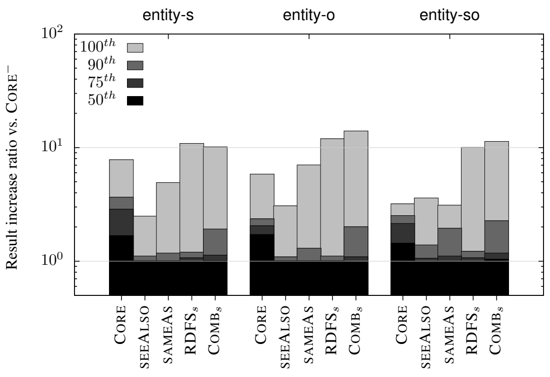

Fig. 9. Percentiles for ratio of increase in runtimes vs. CORE$^-$ for entity-query classes (log)

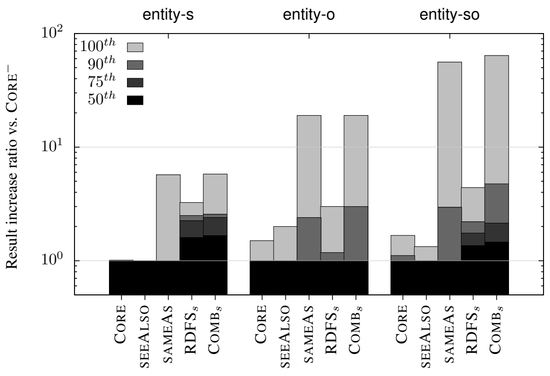

Fig. 10. Percentiles for ratio of increase in results vs. CORE$^-$ for entity-query classes (log)

Conversely, Figure 10 shows that CORE almost never returns additional results beyond CORE$^-$.

Similarly from both figures, we 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 9, we can see that in the majority of cases, other extensions did not affect performance significantly; however, the 90^{th} and 100^{th} percentiles show that reasoning can occasionally increase runtimes by a factor of over ten. However, reasoning can also increase result sizes by a large factor, where modest increases are visible already on the 50^{th} percentile for RDFS$_s$ and COMB$_s$ in the entity-s and entity-so queries: all RDFS rules offer additional data for entity-s* queries, whereas only sub-property reasoning offers additional results for entity-o in the general case. Furthermore, the 75^{th}–

100th percentiles show occasional but very large increases for results in the SAMEAS configuration also.

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

Star-shaped queries are used to display select attributes of a resource useful in a certain context. The results for star-shaped queries follow the same format as before, where Figure 11 shows the average increase in query-time for each configuration over CORE^-, and Figure 12 shows the increase in result size.

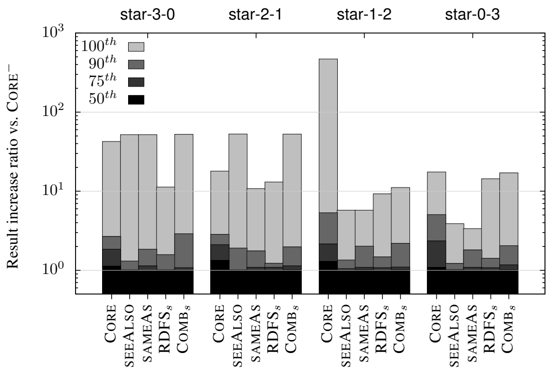

Fig. 11. Percentiles for ratio of increase in runtimes vs. CORE^- for star-query classes (log)

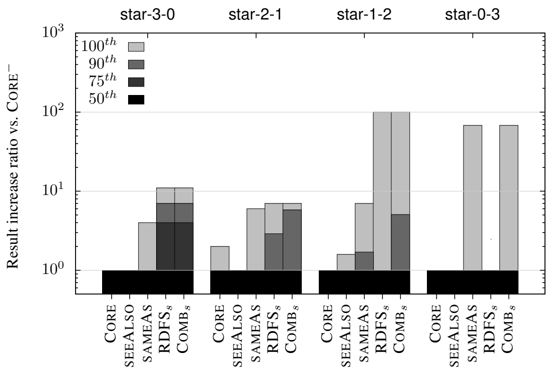

Fig. 12. Percentiles for ratio of increase in results vs. CORE^- for star-query classes (log)

Some similar conclusions can again be drawn as for entity 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 primarily for not dereferencing URIs bound to non-join variables and values for `rdf:type`. The notable outlier for the query class star-1-2 in CORE (on the 100th 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 (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. We also again see that RDFS and `owl:sameAs` reasoning has 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 24), where a small absolute increase could account for a very large relative increase, as per the outliers on the 100th percentile. Also, the large RDFS-related results outlier for the class star-1-2 is attributable 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 discover particular information about neighbouring nodes. When compared with entity 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 the six LiDaQ extensions, we again show the average increase of query time in Figure 13 and the average relative recall improvement in Figure 14. Again, we see the savings in time for selecting CORE^- over CORE, particularly for the o-path-* classes of queries. In general, across all extensions, the performance hits for the 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 100th percentile. With respect to the QWalk results, 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 does find additional results for s-path-* queries, which are notable already on the 75th percentile; this extension performs particularly well for s-path-3 where large gains in results do not cost comparable increases in runtimes. The COMB_s configuration again offers the most results, but—with the exception of CORE—at the cost of the highest runtimes.

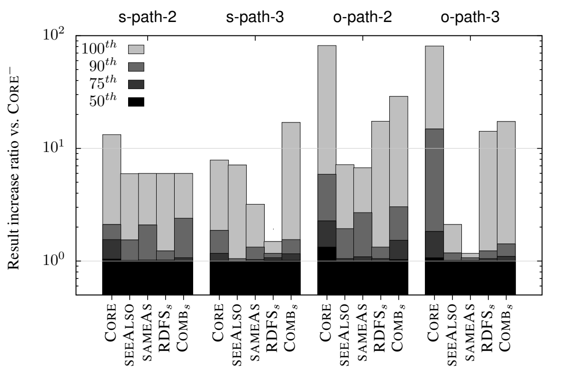

Fig. 13. Percentiles for ratio of increase in runtimes vs. CORE$^-$ for path-query classes (log)

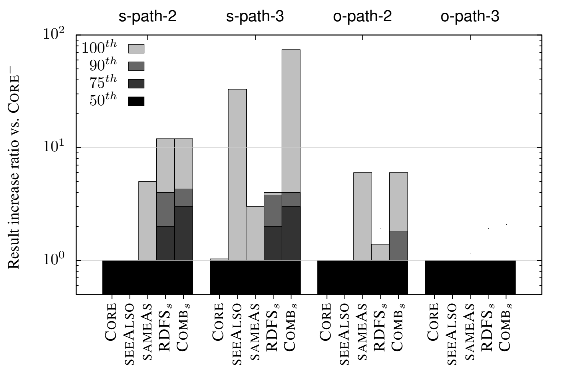

Fig. 14. Percentiles for ratio of increase in results vs. CORE$^-$ for path-query classes (log)

Conclusion Across the hundreds or queries run for the 11 query classes, we consistently find that the CORE$^-$ configuration saves significantly on query runtimes while not significantly reducing result sizes versus CORE. 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 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 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.

To help summarise the potential benefit of each configuration, Table 12 presents the average throughput (results per second) achieved across all queries per query class.[33] 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 entity-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 12

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

	CORE	CORE$^-$	SEEALSO	SAMEAS	RDFS$_s$	COMB$_s$
entity-s	1	1.68	1.67	2.15	1.29	1.53
entity-o	3.97	6.48	6.16	5.7	5.37	4.38
entity-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

9. Conclusion

In theory, proposed link-traversal query approaches for Linked Data have the benefit of up-to-date results and decentralised execution. However, in practice, a thorough evaluation of such methods in realistic uncontrolled environments—for a diverse Web of Data—had not yet been conducted. Herein, we have focused on evaluating LTBQE approaches in this manner, and similarly investigate the possibility of com-

[33]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.

bining lightweight reasoning methods with LTBQE to help squeeze additional answers from query relevant sources, and to help integrate data from diverse data-providers.

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 often touch upon documents from the same domain.

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.* [20]—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.* [18]. 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 [62]. 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 stable experimental queries are available at `http://code.google.com/p/lidaq/wiki/Lidaq`.

Acknowledgements The research presented in this paper has been supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-II).

References

[1] K. Alexander and M. Hausenblas. Describing linked datasets - on the design and usage of voiD, the 'vocabulary of interlinked datasets. In *Linked Data on the Web Workshop (LDOW 09)*, 2009.
[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] T. Berners-Lee. Linked Data. Design issues, W3C, 2006.
[5] T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Jan. 2005. `http://tools.ietf.org/html/rfc3986`.
[6] P. A. Bernstein, M. L. Brodie, S. Ceri, D. J. DeWitt, M. J. Franklin, H. Garcia-Molina, J. Gray, G. Held, J. M. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. F. Naughton, H. Pirahesh, M. Stonebraker, and J. D. Ullman. The Asilomar Report on database research. *SIGMOD Record*, 27(4):74–80, 1998.
[7] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. FactForge: A fast track to the Web of Data. *Sem. Web J.*, 2011.
[8] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. OWLIM: A family of scalable semantic repositories. *SWJ*, 2011.
[9] C. Bizer, R. Cyganiak, and T. Heath. How to publish Linked Data on the Web. `linkeddata.org` Tutorial, July 2008.
[10] C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web & Information Systems*, 2009.
[11] 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.
[12] 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.
[13] G. Cheng and Y. Qu. Term dependence on the Semantic Web. In *International Semantic Web Conference*, pages 665–680, 2008.
[14] R. Cyganiak, A. Hogan, and A. Harth. N-Quads: Extending N-Triples with context. http://sw.deri.org/2008/07/n-quads/.
[15] R. Delbru, G. Tummarello, and A. Polleres. Context-dependent OWL reasoning in Sindice - experiences and lessons learnt. In *RR*, pages 46–60, 2011.
[16] 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 *International Semantic Web Conference (1)*, pages 145–160, 2010.
[17] O. Erling and I. Mikhailov. RDF support in the Virtuoso DBMS. In *Networked Knowledge – Networked Media*. Springer, 2009.
[18] 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.
[19] M. A. Gallego, J. D. Fernández, M. A. Martínez-Prieto, and P. De La Fuente. An empirical study of real-world SPARQL queries. *Challenge*, pages 3–6, 2011.
[20] B. Glimm, A. Hogan, and M. Krötzsch. OWL: Yet to arrive on the Web of Data? In *Linked Data on the Web Workshop (at WWW2012)*, 2012.
[21] O. Görlitz and S. Staab. SPLENDID: SPARQL endpoint federation exploiting voiD descriptions. In *Proceedings of the 2nd International Workshop on Consuming Linked Data*, Bonn,

Germany, 2011.

[22] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics Science Services and Agents on the World Wide Web*, 2005.

[23] 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.

[24] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C Working Draft, Jan. 2012. `http://www.w3.org/TR/sparql11-query/`.

[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 at WWW*, 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 at WWW*, 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] G. Ladwig and A. Harth. CumulusRDF: Linked Data management on nested key-value stores. In *Scalable Semantic Web Systems (SSWS) Workshop (at ISWC2011)*, 2011.

[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 and W. Wöß. RDFStats - an extensible RDF statistics generator and library. In *DEXA Workshops*, pages 79–83, 2009.

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

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

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

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

[46] 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.

[47] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *International Semantic Web Conference*, pages 30–43, 2006.

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

[49] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, Jan. 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[50] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, 2008.

[51] 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.

[52] M. Schmidt, T. Hornung, N. Kuchlin, G. Lausen, and C. Pinkel. An experimental comparison of RDF data management approaches in a SPARQL benchmark scenario. *The Semantic WebISWC 2008*, pages 82–97, 2010.

[53] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SPˆ2Bench: A SPARQL performance benchmark. *Data Engineering 2009 ICDE 09 IEEE 25th International Conference on*, 2009.

[54] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: A federation layer for distributed query processing on Linked Open Data. In *ESWC*, 2011.

[55] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on Linked Data. In *Proceedings of the International Semantic Web Conference ISWC 2011*, 2011.

[56] E. Spertus and L. A. Stein. Squeal: a structured query language for the Web. *Computer Networks*, 33(1-6):95–103, 2000.

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

[58] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.

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

[60] J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Improving the recall of live Linked Data querying through reasoning. In *Web Reasoning and Rule Systems (RR)*, Sept. 2012. (to appear).

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

[62] J. Umbrich, M. Karnstedt, A. Hogan, and J. X. Parreira. Freshening up while staying fast: towards hybrid SPARQL queries. In *EKAW*, 2012.

[63] J. Umbrich, B. Villazón-Terrazas, and M. Hausenblas. Dataset dynamics compendium: A comparative study. In *COLD Workshop*, 2010.

[64] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using MapReduce. In *International Semantic Web Conference*, pages 634–649, 2009.

[65] D. Vrandečíc, M. Krötzsch, S. Rudolph, and U. Lösch. Leveraging non-lexical knowledge for the Linked Open Data Web. *Review of April Fool's day Transactions (RAFT)*, 5:18–27, 2010.

[66] J. Weaver and J. A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In *International Semantic Web Conference*, pages 682–697, 2009.

[67] G. T. Williams. SPARQL 1.1 service description. W3C Working Draft, Jan. 2012. http://www.w3.org/TR/sparql11-service-description/.

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 8.3. We also show results for the SQUIN library: 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. Note that we do not have measurements for the 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 .
}
```

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, particularly for abbreviations). As such, we see that the overall time taken for baseline LiDaQ methods is roughly a function of the politeness policy (two lookups per second) and the number of HTTP lookups required: the query times of

Table 13
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	—	—

over 5 minutes are attributable to the high number of sources that must be accessed. Conversely, although Squin performs more lookups than, *e.g.*, Core$^-$ and Core, and generates the same results, it is much faster, but only by performing at least eight HTTP lookups per second to `data.semanticweb.org` which is four times more than the bounds of our politeness policy.

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.[34]

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 14 tell a different story. The results are the same for all configurations: none of the extensions find any additional results in this case, though they do add an additional 10 seconds to the results. In

[34]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 14

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	—	—

fact, given that SAMEAS and RDFS$_s$ retrieve the same number of sources as CORE$^-$, this result gives us an insight into the local overhead of reasoning, which we see has little effect on query times.

The most striking observation is the source-selection savings for CORE$^-$ vs. CORE, 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, translating into a major time saving. We also note that SQUIN performs fewer lookups than 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 15

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 to 288 in LD1 to 54 in LD3. We can see in Table 15 that for configurations without reasoning, LD3 returns $\sim$57% of the number of results of LD1: the decrease in authors is partially balanced by the addition of another variable in the results. CORE$^-$ offers a moderate performance improvement over CORE while returning the same results. RDFS reasoning increases result sizes for similar reasons as before, 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 .
  ?paper swrc:author ?p .
  ?paper swc:isPartOf ?proceedings .
  ?proceedings swc:relatedToEvent swEswc10: .
}
```

Table 16

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 16, even for CORE$^-$, the query processor performed over 1,800 lookups and our source selection approach does not affect 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, approximately $\frac{1{,}800}{2}$ seconds). 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 17

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`. Although sameAs runs through (taking 3.28× 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 18

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 18, 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 [35], where many triples (including inlinks) were omitted

[35] Last accessed on 2012/02/28.

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. [36] 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 offer 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`[37] 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 the GeoNames domain only contain triples where the URI in question appears in the subject position, not "inlinks". If not for these two issues, we would expect this query to be straightforward for LiDaQ/SQUIN to run.

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 .
}
```

From the results in Table 19, we see the improvements of CORE vs. CORE$^-$. Most prominently however, 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 massive problems, 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].[38] 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 `OutOfMemoryExceptions`.

Table 19
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	—	—

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 20
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	—	—

[36]Last accessed on 2012/08/09.
[37]http://sws.geonames.org/robots.txt

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

As we can see from the results in Table 20, 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, which have since been fixed.[39] 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)

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

Table 21
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 aims to combine data from the `dbpedia.org` domains and the `data.nytimes.com` domain. We already explained that we changed the query predicate `skos:subject` to `dcterms:subject` in order to reflect changes in the DBpedia data model. 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 order of the query pattern. This fact is reflected in the results shown in Table 21, where we only find results if `owl:sameAs` inferencing is enabled; in fact, both configurations which returned results timed out doing so, and again, both configurations involving the dynamic import of schema data threw exceptions.

[39]The document URL in question—`http://dbpedia.org/data/Brazil_national_football_team.xml`—was tested with the W3C RDF/XML validator.

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 22
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	—	—

Table 22 shows that this query involves the largest amount of results and source lookups of all the FedBench queries.[40] 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 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

With respect to the detailed average measures for the QWalk experiments, Table 23 presents the results for entity-* queries, Table 24 gives results for star-* queries, and Table 25 gives results for *-path-* queries. For space reasons, we only present standard deviations for terms, results and time.

[40]We remark that the public centralised SPARQL endpoint for DBpedia often times-out with a `509` response code for this query.

Table 23
Detailed QWalk results for entity-* queries

	Setup	Term		Results		Time (s)		First (s)	HTTP	Data	Inferred
		avg.	σ	*avg.*	σ	*avg.*	σ				
entity-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
entity-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
entity-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

Table 24
Detailed QWalk results for star-* queries

	Setup	Term		Results		Time (s)		First (s)	HTTP	Data	Inferred
		avg.	σ	*avg.*	σ	*avg.*	σ				
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

Table 25
Detailed QWalk results for *-path-* queries

	Setup	Term		Results		Time (s)		First (s)	HTTP	Data	Inferred
		avg.	σ	*avg.*	σ	*avg.*	σ				
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

C. Prefixes (Table 26)

Table 26
Mappings for all prefixes used

Prefix	URI
cb:	http://www.bizer.de#
cbDoc:	http://www4.wiwiss.fu-berlin.de/bizer/foaf.rdf
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