Improving the Recall of Live Linked Data Querying through Reasoning

Jürgen Umbrich[1], Aidan Hogan[1], Axel Polleres[2], Stefan Decker[1]

[1] Digital Enterprise Research Institute, National University of Ireland, Galway
[2] Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria
Emails: {*firstname.lastname*}@deri.org, axel.polleres@siemens.com

Abstract. Linked Data principles allow for processing SPARQL queries on-the-fly by dereferencing URIs. Link-traversal query approaches for Linked Data have the benefit of up-to-date results and decentralised execution, but operate only on explicit data from dereferenced documents, affecting recall. In this paper, we show how inferable knowledge—specifically that found through `owl:sameAs` and RDFS reasoning—can improve recall in this setting. We first analyse a corpus featuring 7 million Linked Data sources and 2.1 billion quadruples: we (1) measure expected recall by only considering dereferenceable information, (2) measure the improvement in recall given by considering `rdfs:seeAlso` links as previous proposals did. We further propose and measure the impact of additionally considering (3) `owl:sameAs` links, and (4) applying lightweight RDFS reasoning for finding more results, relying on static schema information. We evaluate different configurations for live queries covering different shapes and domains, generated from random walks over our corpus.

1 Introduction

Recently, a rich lode of RDF data has been published on the Web as *Linked Data* by governments, academia, industry, communities and individuals alike [15]. Publishing Linked Data is governed by four principles, here summarising [2]: (P1) *use URIs to name things*, such that (P2) those *URIs can be dereferenced via HTTP*, such that (P3) *dereferencing yields useful RDF content* about that which is named, such that (P4) the returned *content includes links* (mentions external URIs) for further discovery. Given that the URIs used to name resources map (through HTTP) to the physical location of structured information about them, information published as Linked Data can be viewed as forming a scale-free, decentralised database, consisting of millions of structured Web documents [13]. Further still, thanks to the provision of typed "RDF links" between such documents [15, § 4.5], agents can traverse and navigate the resulting *Web of Data* in a manner analogous to browsing through the *Web of Documents*.

Tangentially, SPARQL [22]—the W3C standardised RDF query language—provides the declarative means to formulate structured queries against these data. Traditional approaches for posing queries against Linked Data retrieve and cache data in local indexes; however, the dynamicity and scope of Web data

implies that results are often stale or missing. Hartig et al. [12] propose using dereferenceable URIs in a SPARQL query—and recursively, in the intermediate results—to automatically determine a focussed set of sources that, by Linked Data principles, are likely to be *query relevant*, retrieving them live from the Web at query-time and processing them for answers. By operating over compliant Linked Data, their approach bypasses the need for source graphs to be explicitly named or pre-indexed, allowing for *ad hoc*, live discovery. Later work [10] calls this approach *Link Traversal Based Query Execution* (LTBQE). A core challenge for LTBQE is to identify, retrieve and process a minimal number of sources that yield maximal results, keeping response times low while maximising answers.

In this paper, we first reintroduce the LTBQE approach, and highlight the core assumptions under which it operates well. We then show to what extent these assumptions hold in practice—*i.e.*, measuring how much data is attainable from dereferencing—through empirical analysis of a corpus of ∼7.4 m RDF Web documents. We further propose extensions of LTBQE to (i) minimise the number of sources accessed by being more selective about links followed; (ii) increase recall by considering some lightweight semantics of Linked Data that allow for (ii.a) finding additional query-relevant sources and data through consideration of `owl:sameAs` links, and (ii.b) finding additional query-relevant data through rule-based materialisation with respect to a lightweight subset of RDFS (viz. ρDF [20]). We measure the expected effect on recall for each of these extensions through similar analysis of our data. Finally, we generate a diverse set of benchmark queries from our corpus and run them live over remote sources, comparing different LTBQE configurations and extensions.

2 Background and Related Work

Traditional approaches to query Linked Data locally replicate the content of remote Linked Data sources and execute SPARQL queries over the local copy. In previous years, we supported such a service powered by YARS2 [9] allowing for querying over millions of RDF Web documents (and their entailments), but discontinued the endpoint due to prohibitive running costs. Current centralised SPARQL endpoints harvesting Linked Data include "FactForge" [3][3] (powered by BigOWLIM [4]), OpenLink's LOD cache[4] and Sindice's "Semantic Web Index" [21][5] (both powered by Virtuoso [7]). The primary targets for these engines are (i) to have a broad coverage of the Web of Data, (ii) to keep results up to date, (iii) to have fast response times. These objectives are (partially) met using distribution techniques, replication, optimised indexes, compression techniques, data synchronisation, and so forth [4, 7, 9, 21]. However, maintaining a broad, up-to-date and optimised local index is a Sisyphean task.

Federated SPARQL engines execute queries over a group of independent endpoints, dividing and routing sub-queries to individual endpoints [1, 23, 24].

[3] http://factforge.net/sparql
[4] http://lod.openlinksw.com/sparql
[5] http://sparql.sindice.com/

Given the recent spread of SPARQL endpoints on the Web of Data, federation is a timely topic and enjoys increasing attention. However, our techniques operate over raw source documents, not SPARQL endpoints.

Recently, various authors have proposed methods for performing live querying, accessing remote data at runtime. Ladwig and Tran [17] categorise these approaches as follows: (i) top-down query evaluation, (ii) bottom-up query evaluation, and (iii) mixed strategy query evaluation. Top-down evaluation determines remote, query-relevant sources using a *source-selection index*: a local repository summarising information about sources that can vary from inverted-index structures [19, 21], to query-routing indexes [26], schema-level indexes [25], or lightweight hash-based structures [27]. The bottom-up query evaluation strategy involves discovering relevant sources on-the-fly during the evaluation of queries by selectively and recursively following links starting from a "seed set" of URIs taken from the query [12]. The third strategy uses (in a top-down fashion) some knowledge about sources to generate the seed list, then discovering additional relevant sources using a bottom-up approach [17]. All such approaches rely on time-consuming remote lookups, but conversely offer fresh results.

An appealing use-case for live querying is to combine both centralised and live querying results: 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.*, [13, 18, 28]).

3 Preliminaries

We now present some preliminaries. Before we continue, we introduce a motivating example that will be used to explain the concepts involved: Figure 1 illustrates an RDF (sub)graph taken from four (real) interlinked sources on the Web of Data. The graph contains structured information about one publication (`dblpPub:HartigBF09`), "four" people (`oh:olaf`, `cb:chris`, `dblpAuth:Olaf_Hartig`, `dblpAuth:Christain_Bizer`) and four dereferenceable documents.

Presented below Figure 1 are three example queries. For Query 1, the LTBQE approach dereferences `oh:olaf`, finds `cb:chris` as a binding, and dereferences it to look for depictions; however the URI does not dereference, and so the LTBQE resorts to following the `rdfs:seeAlso` link, as supported in the original proposal [12]. For Query 2, `oh:olaf` is again dereferenced, the FOAF file of `cb:chris` is found through a see-also link; however, to traverse further and find answers, the query-processor needs to traverse the `owl:sameAs` link and also support the semantics thereof. We propose and evaluate this extension later. Finally, the FOAF vocabulary defines `foaf:name` to be a sub-property of `rdfs:label`, where RDFS reasoning is required to answer Query 3; we also propose this extension.

We now formally define these concepts, covering preliminaries relating to RDF and Linked Data (§ 3.1), SPARQL (§ 3.2) and RDFS & OWL (§ 3.3).

3.1 RDF and Linked Data

We first provide some notation for dealing with RDF and Linked Data principles.

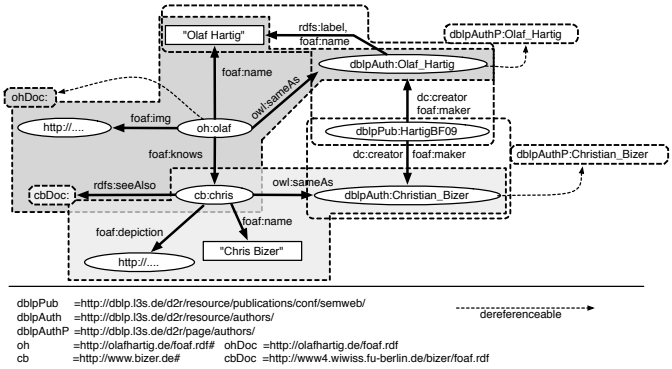


Fig. 1: Snapshot of a sub-graph from the Linked Open Data Web

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

Query 1: Friends' images

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

Query 2: Coauthors

```
SELECT ?f ?l WHERE {
    oh:olaf foaf:knows ?f .
    ?f rdfs:label ?l }
```

Query 3: Friends' labels

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* $\mathbf{L}$*. 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). A 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 resp. triple position (*terms *gives all positions).*

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 that 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* $(u, \mathsf{get}(u))$*, and* $\mathsf{merge}(\Gamma) := \biguplus_{(u,G)\in\Gamma} G$ *as the RDF merge of graphs in* Γ*, which preserves the uniqueness of blank-node labels across graphs [14].*

Definition 3 (Dereferencing RDF). *A URI may issue a HTTP redirect to another URI with a* 30x *response code; we denote this function as* $\mathsf{redir} : \mathbf{U} \to \mathbf{U}$*, which strips the fragment identifier of a URI (if present) and which would (thereafter) map a URI 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 (as long as it returns* some *valid RDF).*

Taking Figure 1, *e.g.*, $\mathsf{redir}(\texttt{oh:olaf}) = \texttt{ohDoc:}$, $\mathsf{deref}(\texttt{oh:olaf}) = \mathsf{deref}(\texttt{ohDoc:}) = \{(\texttt{oh:olaf}, \texttt{foaf:name}, \texttt{"Olaf Hartig"}) \ldots\}$ and $\mathsf{deref}(\texttt{cb:chris}) = \emptyset$.

3.2 SPARQL

We now introduce some concepts relating to SPARQL [22]. Note that herein, we focus on evaluating simple, conjunctive, *basic graph patterns* (BGPs) .

Definition 4 (Variables, Triple Patterns and Queries (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 replaced with variables). 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 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*, which binds variables in* $\mathrm{dom}(\mu) \subset \mathbf{V}$ *to* $\mathbf{UBL}$ *and which 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* $\mu(Q) := \{\mu(tp) \mid tp \in Q\}$*. Now, we define the set of solutions for a query* Q *over a Linked Dataset* Γ *as* $\Omega(\Gamma, Q) := \{\mu \mid \mu(Q) \subseteq \mathsf{merge}(\Gamma) \wedge \mathrm{dom}(\mu) = \mathsf{vars}(Q)\}$*. Note that herein, and unlike SPARQL, solutions are given as sets (not multi-sets), implying a default* DISTINCT *semantics for queries.*

Taking an example, if we let Γ be Figure 1 and Q be Query 3, then $\Omega(\Gamma, Q) = \{(\texttt{?f}, \texttt{cb:chris}), (\texttt{?l}, \texttt{"Chris Bizer"})\}$.

3.3 RDFS and OWL

We define some preliminaries relating to RDFS and OWL. In particular, we support a miniature subset of OWL 2 RL/RDF rules for supporting `owl:sameAs` entailments, given in Table 1. Our RDFS rules are the subset of ρDF rules proposed by Muñoz et al. [20], which deal with instance data entailments.[6] 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 [5, §4.2, §6]. The rules we consider are given in Table 1.

[6] We drop *implicit typing* [20] rules but allow generalised RDF in interim inferences.

Table 1: ρDF and owl:sameAs rules with OWL 2 RL/RDF naming

ID	Body	Head
PRP-SPO1	?p_1 rdfs:subPropertyOf ?p_2 . ?s ?p_1 ?o .	?s ?p_2 ?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	?c_1 rdfs:subClassOf ?c_2 . ?s a ?c_1 .	?s a ?c_2 .
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′ .

Definition 6 (Entailment Rules and Closure). *Given a ruleset R and a Linked Dataset Γ, we denote by ${}^{R}\Gamma := \Gamma \cup (\upsilon, G)$ the closure of Γ wrt. R, where (abusing notation) G contains the materialised inferences from recursively applying the rules in R over $\mathsf{merge}(\Gamma) \uplus G$ up to a fixpoint, and where υ is a built-in URI for naming the materialised graph.*

4 Link Traversal Based Query Execution

We first introduce the Link Traversal Based Query Execution (LTBQE) approach introduced by Hartig et al. [12] (§ 4.1), and then look at extensions of the approach (§ 4.2). Our formalisms are tailored for the purpose of this work; a comprehensive study of semantics and computability is presented in [11].

4.1 Baseline LTBQE

Definition 7 (LTBQE Query Relevant Sources and Answers). *Define $\mathsf{derefs} : 2^{\mathbf{U}} \to \mathbf{U} \times 2^{\mathbf{G}}$; $U \mapsto \{(\mathsf{redirs}(u), \mathsf{deref}(u)) \mid u \in U)\}$ as the mapping from a set of URIs to the Linked Dataset it represents by dereferencing all URIs. Given a BGP query Q as before, let $U_Q := \mathsf{terms}(Q) \cap \mathbf{U}$ denote the set of URIs appearing in Q. Let $\Gamma_0^Q := \mathsf{derefs}(U_Q)$ represent the dataset retrieved by dereferencing all query URIs.*[7] *Next 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, and let $U_i := \{u \in \mathsf{uris}(\mu) \mid \exists \mu, \exists tp \in Q \text{ s.t. } \mu(\{tp\}) \subseteq \mathsf{merge}(\Gamma_{i-1}^Q)\}$ for $i \in \mathbb{N}$ be the set of URIs that appear as a solution mapping for a triple pattern in Q for the dataset Γ_{i-1}^Q, and let $\Gamma_i^Q := \mathsf{derefs}(U_i) \cup \Gamma_0^Q$.*[8] *The set of* LTBQE query relevant sources *for Q is given as the least n such that $\Gamma_n^Q = \Gamma_{n+1}^Q$, denoted simply Γ^Q. The set of* LTBQE query answers *for Q is given as $\Omega(\Gamma^Q, Q)$, or simply Ω^Q.*

With regards to completeness, let get denote the dataset (theoretically) represented by the entire Web of Data (note: $\mathsf{get} \subset \mathbf{U} \times 2^G$). One may then ask when Ω^Q is complete with respect to get. A trivial sufficient condition for completeness

[7] One could consider Γ_0^Q as also containing "seed" data [12].
[8] Or, equivalently (for static data) $\Gamma_i^Q := \Gamma_{i-1}^Q \cup \mathsf{derefs}(U_i \setminus U_{i-1})$.

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

Query 4: Not dereferenceable

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

Query 5: Connected by literal

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

Query 6: Not in dereferenceable document

```
SELECT ?paper WHERE {
  cb:chris owl:sameAs ?dblpC .
  oh:olaf owl:sameAs ?dblpO .
  ?dblpC foaf:maker ?paper .
  ?dblpO foaf:maker ?paper .
}
```

Query 7: Query answer only reachable from the seed URI `oh:olaf`

is given by $\Gamma^Q = \mathsf{get}$; such a case is, however, infeasible. Otherwise the completeness condition is rather simple and entirely unverifiable: Γ^Q must contain all of the data relevant on the Web to answer the query. Of course, verifying that the condition does not hold is significantly easier in many cases. The implications are that: first, given a query with no dereferenceable URIs, LTBQE cannot return results (as per Query 4 where `cb:chris` does not dereference). Second, given a query with multiple URIs, different reachability conditions can occur from different starting points (as per Query 7 where the answer is only reachable starting from `ol:olaf`); thus, all query URIs must be initially retrieved [12]. Third, answers "connected" by literals or involving blank-nodes in unreachable documents will often affect completeness (as per Query 5 where the answer is connected by the literal `"Olaf Hartig"`', here not yet considering `owl:sameAs`). Fourth, in the general case, reachability is heavily dependent on the amount of data returned by the $\mathsf{deref}(u)$ function, which would ideally return all triples mentioning u on the Web of Data (*e.g.*, in Query 6, the `owl:sameAs` inlinks for `dblpAuth:Olaf_Hartig` are not in its dereferenced document and will not be found). The fourth assumption is clearly idealised; hence, in Section 5 we will empirically analyse how much the assumption holds in practice, giving insights into the recall of LTBQE. First, however, we propose our reasoning extensions.

4.2 Extending LTBQE

1. Following `rdfs:seeAlso`: The first extension to LTBQE is proposed by Hartig et al., and uses `rdfs:seeAlso` links to extend the set of query sources. Adapting Definition 7, let $\bar{\Gamma}_0^Q := \Gamma_0^Q$ and let:

$$\bar{U}_i := U_i \cup \{u \in \mathbf{U} \mid \exists u' \in U_i \text{ s.t. } (u', \texttt{rdfs:seeAlso}, u) \in \mathsf{merge}(\bar{\Gamma}_{i-1}^Q)\}\,,$$

let $\bar{\Gamma}_i^Q := \mathsf{derefs}(\bar{U}_i) \cup \Gamma_0^Q$, and finally let $\bar{\Gamma}^Q$ be the fixpoint as before and let $\bar{\Omega}^Q$ be the respective solutions. This extends LTBQE to find more sources through `rdfs:seeAlso` links. An example for this has been presented in Query 1.

2. Following and reasoning over `owl:sameAs`: We propose an extension of LTBQE to consider `owl:sameAs` inferences. Let R denote the set of rules of the

form EQ-* in Table 1. Let now ${}^{e}\Gamma_0^Q := {}^{R}\Gamma_0^Q$ (recalling ${}^{R}\Gamma$ from Def. 6 and Γ_0^Q from Def. 7), and let:

$$U'_i := \{u \in \mathsf{uris}(\mu) \mid \exists \mu, \exists tp \in Q \text{ s.t. } \mu(\{tp\}) \subseteq \mathsf{merge}({}^{e}\Gamma_{i-1}^Q)\}\,,$$
$${}^{e}U_i := \{u \in \mathbf{U} \mid \exists u' \in U'_i \text{ s.t. } (u', \texttt{owl:sameAs}, u) \in \mathsf{merge}({}^{e}\Gamma_{i-1}^Q)\}\,,$$

where ${}^{e}\Gamma_i^Q := {}^{R}\mathsf{derefs}({}^{e}U_i) \cup {}^{e}\Gamma_0^Q$, and finally let ${}^{e}\Gamma^Q$ be the fixpoint as before and let ${}^{e}\Omega^Q$ be the respective solutions. Here, `owl:sameAs` links are used to expand the set of query relevant sources, and `owl:sameAs` rules are used to materialise inferable knowledge given by the OWL semantics, potentially generating additional answers. An example for this has been presented in Query 2.

3. Reasoning for ρDF: We propose a final novel extension of LTBQE to consider a subset of RDFS reasoning as per the PRP-* and CAX-SCO rules in Table 1, which we again denote here by R. We currently consider a static set of schema data representing vocabularies on the Web, which we denote by Γ^{voc}. This serves as input into the LTBQE algorithm. In future work, we plan to investigate dereferencing schema knowledge live from the Web of Data.

Now, adapting Definition 7, let ${}_{\rho}\Gamma_0^Q := \Gamma^{voc} \cup {}^{R}\Gamma_0^Q$ and let:

$${}_{\rho}U_i := \{u \in \mathsf{uris}(\mu) \mid \exists \mu, \exists tp \in Q \text{ s.t. } \mu(\{tp\}) \subseteq \mathsf{merge}({}_{\rho}\Gamma_{i-1}^Q)\}\,,$$

where ${}_{\rho}\Gamma_i^Q := {}^{R}\mathsf{derefs}({}_{\rho}U_i) \cup {}_{\rho}\Gamma_0^Q$, and finally let ${}_{\rho}\Gamma^Q$ be the fixpoint as before and let ${}_{\rho}\Omega^Q$ be the respective solutions. Here, RDFS rules and background schema knowledge (Γ^{voc}) are used to materialise inferable knowledge, potentially generating additional answers (and thus possibly finding new query relevant sources). An example for this has been presented in Query 3.

Combined Of course, the above methods can be combined in a natural fashion, where, *e.g.*, for combining all extensions, the query relevant sources are denoted ${}^{e}_{\rho}\bar{\Gamma}^Q$ and the answers by ${}^{e}_{\rho}\bar{\Omega}^Q$.

5 Empirical Study

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 those lines, we now present the results of our empirical study of a Linked Data corpus. 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 recall of data considering (1) explicit, dereferenceable information; (2) including `rdfs:seeAlso` links [12]; (3) including `owl:sameAs` links and inferable knowledge; (4) including RDFS reasoning.

Empirical corpus We use the Billion Triple Challenge 2011 dataset for our survey, which was crawled in mid-May 2011 from 7.4 million RDF/XML documents spanning 791 pay-level domains (data providers). The resulting corpus contains 2.15 g quadruples (1.97 g unique triples) mentioning 538 m RDF terms, of which 52 m (10%) are Literals, 382 m (71%) are blank nodes, and 103 m (19%) are URIs. We denote the corpus as $\Gamma_\sim$. It's important to note that this corpus is only a *sample* of the Web of Data; in particular, we only use the information about HTTP lookups provided by the dataset. We found that a total of 25.4 m lookups were performed (excluding `robots.txt`). As such, we only have knowledge of redir and deref functions for 18.65 m 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 m, 8.37 m (44.8%) dereferenced to RDF; we denote these by $D_\sim$. Further note that, wrt. the Web of Data, our sample recall measures specify an upper bound.

RDFS Schema From our corpus, we extract a static set of schema data for the RDFS reasoning. As argued in [6], 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.[9] Thus, we perform *authoritative reasoning*, which conservatively discards certain third-party schema axioms (cf. [6]). Our schema data only considers triples of the following form:

$$(s, \texttt{rdfs:subPropertyOf}, o) \in \mathsf{deref}(s), (s, \texttt{rdfs:subClassOf}, o) \in \mathsf{deref}(s)$$
$$(s, \texttt{rdfs:domain}, o) \in \mathsf{deref}(s), (s, \texttt{rdfs:range}, o) \in \mathsf{deref}(s)$$

We extracted a total of 397 thousand such authoritative RDFS triples from 98 PLDs as follows: 334 thousand `rdfs:subClassOf` (82 PLDs); 11 thousand `rdfs:subPropertyOf` (67 PLDs); 26 thousand `rdfs:domain` (79 PLDs); and 26 thousand `rdfs:range` (77 PLDs).

5.1 Recall for Baseline

We first measure the average dereferenceability of information in our sample. For a dereferenceable uri d, we compute the *sample dereferencing recall* $\mathsf{sdr}(d)$ as the ratio of the number of unique triples mentioning d in $\mathsf{deref}(d)$ vs. unique triples mentioning d across the entire sample. We denote by $\mathsf{sdr}_\sim$ the average $\mathsf{sdr}(d)$ for all $d \in D_\sim$. We also analyse the $\mathsf{sdr}(d)$ restricting the specific triple positions where d appears. We ignore d in the average if it does not appear in the relevant triple position in the sample. Table 2 presents the results for different triple positions. Column *type*-**object** considers d only when appearing as object in a triple with the predicate `rdf:type` (a class position).

The analysis provides some interesting initial insights into the LTBQE approach. Given a HTTP URI without a common *non*-RDF extension, we have a 44.8% success ratio to receive RDF/XML content regardless of the triple position. If such a URI dereferences to RDF, we receive on average (at most) 51%

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

Table 2: Dereferenceability results for different triple positions

Measure	**any**	**subject**	**predicate**	**object**	*type*-**object**
$\lvert U_\sim \rvert$	18.65 m	9.55 m	47.67 k	9.73 m	213.38 k
$\lvert D_\sim \rvert$	8.37 m	8.09 m	745	4.5 m	21.1 k
$\lvert U_\sim \rvert / \lvert D_\sim \rvert$	0.44	0.85	0.01	0.46	0.09
average $\mathtt{sdr}_\sim$	**0.51**	**0.95**	**0.00007**	**0.438**	**0.002**
std. dev. $\pm$ $\mathtt{sdr}_\sim$	0.5	$\pm$0.195	$\pm$0.008	$\pm$0.458	$\pm$0.05

of all triples in which it appears on the Web. Given a pattern with a URI in the subject position, the dereferenceable ratio increases to 95%; for objects, the ratio drops to 43.8%; LTBQE would perform poorly for triple patterns with (only) a URI in the predicate position (0.007%); *etc.* High standard deviations imply that the dereferenceability is often "all or nothing". In summary, LTBQE performs well when URIs appear as the subject of triple patterns, moderately when URIs appear in the object, but will perform poorly when URIs appear in the predicate or object of an `rdf:type` triple. In practice, documents dereferenced by property and class terms do not host a high percentage of their extension.

5.2 Recall for Extensions

We now measure the `sdr` increase given by extending LTBQE to also consider `rdfs:seeAlso` and `owl:sameAs` links, as well as inferable knowledge given by `owl:sameAs` and RDFS reasoning.

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 2% (201 k URIs). Where such links exist, following them increases the amount of unique triples by a factor of 1.006× vs. triples in the dereferenced document alone. We conclude that, in the general case, considering `rdfs:seeAlso` triples will only marginally affect the recall increase of LTBQE.

Benefit of following `owl:sameAs` *links & inferable knowledge* We measured the percentage of dereferenceable URIs in $D_\sim$ which have at least one `owl:sameAs` link in their dereferenced document to be 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 by a factor of 2.5× vs. the unique (explicit) triples in the dereferenced document alone. We conclude that, in the general case, `owl:sameAs` links are only sometimes found for dereferenceable URIs, but where available, following them and applying entailment generates significantly more data for generating answers (albeit potentially producing "duplicate" answers under different URI aliases).

Benefit of including ρDF inferable knowledge We measured the percentage of dereferenceable URIs in $D_{\sim}$ whose dereferenced documents given non-empty unique entailments through authoritative ρDF reasoning with respect to Γ^{voc} as 81%. Where such entailments are non-empty, they increase the amount of unique triples by a factor of 1.78× vs. the unique (explicit) triples in the dereferenced document. We conclude that such reasoning often increases the amount of data available for LTBQE query answering, and by a significant amount.

6 Evaluation

In order to test our methods for queries covering a diverse set of sources and query types, we evaluate our proposed LTBQE extensions for a set of pseudo-randomly generated queries extracted from our Linked Data corpus. These queries are then applied live to determine real-world behaviour. Our aim is to compare and contrast different setups and assess our proposed extensions in a realistic scenario. To the best of our knowledge, we are the first work to evaluate the original LTBQE proposal in a live and diverse environment.

Implementation: We have (re-)implemented Hartig et al.'s iterator-based algorithm for LTBQE (which was shown to be complete) [12]. We use ARQ to parse and process input SPARQL queries.[10] We further use the LDSpider crawling framework for performing live Linked Data lookups; 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.[11] We use the SAOR engine to support the aforementioned rule-based reasoning extensions [6]. Note that we use the same input RDFS data as used in the empirical study of the previous section.

Optimised LTBQE: Inspired by our empirical analysis, we also implement and evaluate a variation of the LTBQE approach which does not dereference URIs appearing only in the predicate position of a (possibly partially bound) triple-pattern. Further, we add another optimisation to avoid dereferencing URIs that are only bound by non-distinguished variables not appearing elsewhere in the query (*i.e.*, variables whose value is not used elsewhere). Since these optimisation reduce the number of query-relevant sources, they may in theory lead to less results, though in practice (and as per our empirical survey), we would expect a minimal change in recall over the baseline.

Evaluation Queries: We benchmark queries of elemental graph shapes, *viz.,* entity, star and path queries.
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

[10] http://jena.sourceforge.net/ARQ/
[11] http://code.google.com/p/ldspider/

the subject (*entity-s*); as the object (*entity-o*); as the subject *and* object (*entity-so*). An example for *entity-so* would be `{<d> ?p1 ?o . ?s ?p2 <d> .}`. These type of queries are very common in Linked Data browsers or display interfaces.
Star queries (*star-*[*s3*|*o3*|*s1-o1*|*s2-o1*|*s1-o2*]) contain three acyclic triple patterns which share exactly one URI (called the centre node) where predicate terms are constant. We generate four variations of such queries, differing in the number of triple patterns where the centre node appears as the subject (*s*) or object (*o*). An example for *star-s2-o1* would be `{<d> foaf:knows ?o ; foaf:name ?o1 . ?o3 dc:creator <d> .}`
Path queries ([*s*|*o*]-*path*-[*2*|*3*]) consist of 2 or 3 triple patterns that form a path where 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. We generate four path sub-types: path shaped queries of length 2 and 3 where either the subject or object of one triple pattern is a constant. An example for *s-path-2* is Query 1.

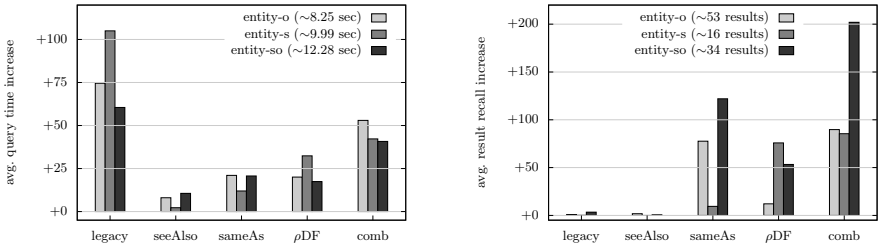
Query generation: In total, we generate 100 `SELECT DISTINCT` queries for *each* of the above 11 query shapes using random walks in our corpus. To help ensure that queries return non-empty results (in case there are no HTTP connection errors or time outs) we consider dereferenceable information for the query generation which (1) picks randomly a pay-level-domain available in the dereferenceable URIs $D_{\sim}$, (2) selects randomly a URI from $D_{\sim}$ for that PLD and (3) generates appropriate triple patterns from the dereferenceable document of the selected URI. For path shaped queries, when performing steps (2) and (3), the URI for the next triple pattern is selected out of the URIs contained for the previous triple pattern, as per a random walk of dereferenceable URIs. Distinguished variables are picked by randomly choosing a single variable as distinguished and make further variables distinguished with a probability of 0.5.

Benchmark Stable Queries: We observed that the results for the same query varied over different runs and thus introduce the notion of "benchmark stable queries" which are queries for which the response codes for the baseline URIs are the same across all setups runs. The variation of results for different runs is caused by remote server or connection failures while dereferencing content, *e.g.*, we sometimes encountered `503 - Service unavailable` response codes possibly due to temporarily high loads on remote servers. Only considering stable queries improves the comparability of results across different setups. Table 3 shows that in average ∼94% ($\frac{1,029}{1,100}$) of the queries are stable from which ∼38% ($\frac{389}{1,029}$) returned empty results. We inspected the causes for the empty result set for all "stable" queries (right side of Table 3). The typewritten numbers correspond to HTTP server response codes.[12] The column "*mixed*" indicates that there are at least two URIs with different response codes and the column "*data*" indicates that the

[12] Aside from common response codes, a `498` code denotes robots.txt forbidden access, `499` denotes that a server returned a mime type different to `application/rdf+xml`, `602` denotes socket timeouts and `603` denotes unknown host exceptions.

Table 3: Statistics about stability of queries.

Breakdown (1,100)			Causes for empty results (389)									
stable	*(empty)*	*unstable*	403	404	498	499	500	502	602	603	*mixed*	*data*
1,029	(389)	71	15	110	15	38	18	2	6	85	7	93



(a) Average *time* increase.

(b) Average *recall* increase.

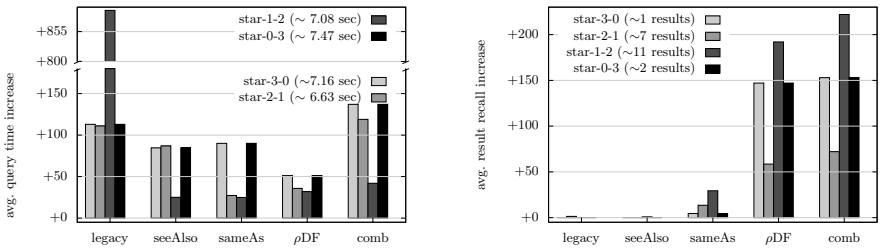
Fig. 2: Average *time* and *recall* increase relative to baseline for *entity queries*.

missing results are not HTTP-related (*e.g.*, the data changed). The main reasons for empty "stable" queries are (i) either query relevant documents are not available any more (404), (ii) the IP address of a host could not be determined (indicated by 603) or (iii) the underlying data changed (last column).

Results per Query Class: We execute each query with six different setups: legacy denotes the original LTBQE approach; base denotes optimised LTBQE; select denotes seeAlso, sameAs, and ρDF which all extend base; comb denotes all extensions of base applied together. We discuss the results for each query-type, presented as bar plots showing the average recall and time increase per setup vs. base (represented as the x-axis). For ρDF, we do not include the time needed to load schema data, which can be done prior to query time.

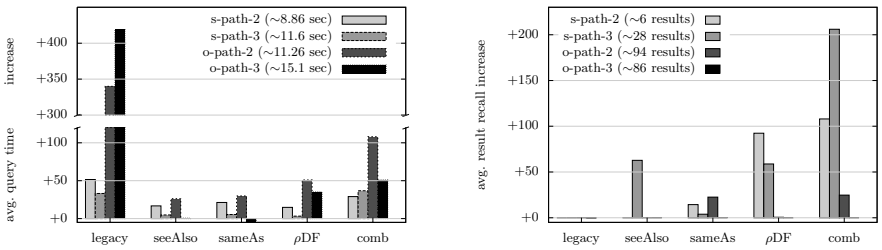
Entity queries Figure 2a shows that the original source selection approach requires between 60% to 110% additional time compared to our source selection optimisation and increases the recall by 3%, shown in Figure 2b. *All such figures include the absolute baseline results (represented by the x-axis) in the legend.* Our extensions increase the recall for all entity queries with a maximum increase of 200% for *entity-so* queries and the combination of all extensions. To enable such increases in recall, the query time increases by up to 55%.

Star queries The results for the star shaped queries, presented in Figure 3a and Figure 3b show similarities with the observations for the entity queries. The query times increase without an improvement of the recall if we dereference all appearing URIs and our extensions improve the recall by a maximum of 140% for 3 out of the 4 query classes. It is worth noting the small volume of



(a) Average *time* increase. (b) Average *recall* increase.

Fig. 3: Average *time* and *recall* increase relative to baseline for *star queries*.



(a) Average *time* increase. (b) Average *recall* increase.

Fig. 4: Average *time* and *recall* increase relative to baseline for *path queries*.

results for the baseline, which may skew average relative increases. Further, we observed some extreme outliers for the query class *star-1-2* due to one query which took around 1 hour to terminate because of a document download from the ecowlim.tfri.gov.tw provider (which did not even contribute to the results).

Path queries The results for the path shaped queries show an average time increase of up to 420% vs. `legacy` compared to our optimised selection for the two *o-path* query classes in Figure 4a. In contrast to the previous query classes, we see in Figure 4b that the `seeAlso` extension improves the recall by over 50% for the *s-path-3* query class, which is the highest measured improvement for that extension across all query classes. In addition, we observe a recall increase of at least 50% for *s-path-** queries with the ρDF and `comb` setups, whereas we measured only a marginal increase for *o-path-** queries.

7 Conclusion

Proposed link-traversal query approaches for Linked Data have the benefit of up-to-date results and decentralised execution, but operate over incomplete knowledge available in dereferenced documents, thus affecting recall for results.

We empirically study this issue for a large sampling of the Web of Data, consisting of 7.4 million Linked Data documents and 2.1 billion quadruples. We further propose to improve recall by considering inferable knowledge, specifically that found through `owl:sameAs` and RDFS reasoning. We again validate our extensions by analysis of our corpus, where we show increases in data available to the LTBQE approach (1) of 1.006× considering `rdfs:seeAlso` information as proposed in [12], (2) of 2.5× considering `owl:sameAs` and (3) of 1.8× if we apply ρDF reasoning using static schema information. We generate and run queries (of eleven different shapes) live over the Web of Data, comparing six different setups, demonstrating the degree to which our extensions find additional results at the cost of accessing more sources and thus taking longer. In addition, our comprehensive experiment also highlights the problem of unreliable server behaviour, which affects query processing and is symptomised by outliers in results and unavailability of data for certain queries.

Future Work We plan to extend our entailment rules to cover more of OWL 2 RL/RDF and to investigate changes in recall when considering dynamically dereferenced schema data vs. static schema data. We also plan to use `owl:sameAs` optimisations for canonicalising equivalent URIs as opposed to materialising all equivalent data. Another open issue relates to that of data quality, where, *e.g.*, Halpin et al. [8] suggest that `owl:sameAs` may often be unreliable; we have experiences of dealing with data-quality issues for reasoning in other works [16]. From such work, we herein borrowed the notion of authoritative RDFS reasoning; further measures to make results more "robust" are a subject for future work.

Given the relatively slow response times from the LTBQE approach (where 1% of all queries returned in less than a second, whereas 67% of the queries executed in less than 10 seconds), our ultimate goal is to use such live querying techniques, in a best-effort manner, to compliment centralised query services with fresh answers, particularly for query patterns that are determined to be dynamic and for which data should be retrieved directly from source. In this scenario, our optimisations should help to get faster live-query answers and our extensions to find further answers. In general, combining different Linked Data querying techniques to find a sweet-spot between fast response times and a high recall of (fresh) answers is an open question, the solution to which is likely to be user- and query-specific. Along such lines, in this paper we have formalised, proposed and evaluated the feasibility in a real-world setting of several novel live querying techniques, a selection of which also incorporate lightweight reasoning.

Acknowledgements: *This research has been supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-II).*

References

1. C. B. Aranda, M. Arenas, and Ó. Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *ESWC*, 2011.
2. T. Berners-Lee. Linked Data. Design issues, W3C, 2006.
3. 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.

4. B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. Owlim: A family of scalable semantic repositories. *SWJ*, 2011.
5. C. Bizer, R. Cyganiak, and T. Heath. How to publish Linked Data on the web. `linkeddata.org` Tutorial, July 2008.
6. P. A. Bonatti, A. Hogan, A. Polleres, and L. Sauro. Robust and scalable Linked Data reasoning incorporating provenance and trust annotations. *JWS*, 2011.
7. O. Erling and I. Mikhailov. RDF support in the virtuoso dbms. In *Networked Knowledge – Networked Media*. Springer, 2009.
8. 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.
9. A. Harth, J. Umbrich, A. Hogan, and S. Decker. YARS2: A federated repository for querying graph structured data from the web. In *ISWC*, 2007.
10. O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *ESWC*, 2011.
11. O. Hartig. SPARQL for a web of Linked Data: Semantics and computability. In *ESWC*, 2012.
12. O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL queries over the web of Linked Data. In *ISWC*, 2009.
13. O. Hartig and A. Langegger. A database perspective on consuming Linked Data on the web. *Datenbank-Spektrum*, 2010.
14. P. Hayes. RDF semantics. W3C Recommendation, Feb. 2004.
15. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011.
16. A. Hogan. *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. PhD thesis, DERI, NUIG, 2011.
17. G. Ladwig and T. Tran. Linked Data query processing strategies. In *ISWC*, 2010.
18. G. Ladwig and T. Tran. SIHJoin: Querying remote and local Linked Data. In *ESWC*, 2011.
19. Y. Li and J. Heflin. Using reformulation trees to optimize queries over distributed heterogeneous sources. In *ISWC*, 2010.
20. S. Muñoz, J. Pérez, and C. Gutierrez. Simple and efficient minimal RDFS. *JWS*, 2009.
21. 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.
22. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, Jan. 2008. http://www.w3.org/TR/rdf-sparql-query/.
23. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, 2008.
24. 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.
25. H. Stuckenschmidt, R. Vdovjak, G.-J. Houben, and J. Broekstra. Index structures and algorithms for querying distributed RDF repositories. In *WWW*, 2004.
26. T. Tran, L. Zhang, and R. Studer. Summary models for routing keywords to Linked Data sources. In *ISWC*, 2010.
27. J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over Linked Data. *WWWJ*, 2011.
28. J. Umbrich, M. Karnstedt, A. Hogan, and J. X. Parreira. Freshening up while staying fast: Towards hybrid SPARQL queries. In *EKAW*, 2012.