Walking Without a Map: Ranking-Based Traversal for Querying Linked Data

Olaf Hartig and M. Tamer Ozsu

Conference Publication

Original Publication:

Olaf Hartig and M. Tamer Ozsu, Walking Without a Map: Ranking-Based Traversal for Querying Linked Data, SEMANTIC WEB - ISWC 2016, PT I, 2016. pp.305-324.
http://dx.doi.org/10.1007/978-3-319-46523-4_19

Walking without a Map: Ranking-Based Traversal for Querying Linked Data

Olaf Hartig[1] and M. Tamer Özsu[2]

[1] Department of Computer and Information Science (IDA), Linköping University, Sweden
olaf.hartig@liu.se
[2] Cheriton School of Computer Science, University of Waterloo, Canada
tamer.ozsu@uwaterloo.ca

Abstract The traversal-based approach to execute queries over Linked Data on the WWW fetches data by traversing data links and, thus, is able to make use of up-to-date data from initially unknown data sources. While the downside of this approach is the delay before the query engine completes a query execution, user perceived response time may be improved significantly by returning as many elements of the result set as soon as possible. To this end, the query engine requires a traversal strategy that enables the engine to fetch result-relevant data as early as possible. The challenge for such a strategy is that the query engine does not know a priori which of the data sources discovered during the query execution will contain result-relevant data. In this paper, we investigate 14 different approaches to rank traversal steps and achieve a variety of traversal strategies. We experimentally study their impact on response times and compare them to a baseline that resembles a breadth-first traversal. While our experiments show that some of the approaches can achieve noteworthy improvements over the baseline in a significant number of cases, we also observe that for every approach, there is a non-negligible chance to achieve response times that are worse than the baseline.

1 Introduction

The availability of large amounts of Linked Data on the World Wide Web (WWW) presents an exciting opportunity for building applications that use the data and its cross-dataset connections in innovative ways. This possibility has spawned research interest in approaches to enable such applications to query Linked Data [4,7]. A well-understood approach to this end is to populate a centralized repository of Linked Data copied from the WWW. By using such a repository it is possible to provide almost instant query results. This capability comes at the cost of setting up and maintaining the centralized repository. Further limitations of this approach are that query results may not reflect the most recent status of the copied data, new data and data sources cannot be exploited, and legal issues may prevent storing a copy of some of the data in the repository.

To address these limitations a number of works adopt an alternative view on querying Linked Data: The idea is to conceive the Web of Linked Data itself as a distributed database in which URI lookups are used to access data sources at query execution time [8,11,12,13,16,17]. A particularly interesting approach is *traversal-based query execution* which intertwines the query execution process with a traversal of data links [8,11,12,13]. This approach can discover initially unknown data and data sources on the fly, and it can be used to start querying right away (without first having to populate a repository of data). An inherent downside, however, is the delay before the data

retrieval process terminates and a complete query result can be returned to the user. Nonetheless, users may want to start receiving elements of the query result set as soon as possible. The following example shows that the user experience may be improved significantly by query optimization approaches that aim to reduce the *response times* of query executions, that is, the times required to find a particular number of result elements (as opposed to the overall time required to complete the query execution).

Example 1. Consider the following SPARQL query from the FedBench benchmark [15].

```
SELECT * WHERE { ?person nyt:latest_use ?mentionInNYT .  ?person owl:sameAs ?chancellor .
                 ?chancellor dct:subject <http://dbpedia.org/resource/Category:Chancellors_of_Germany> }
```

We used the URI at the end of this query as a starting point for a traversal-based execution of the query over the WWW (under c_{Match}-bag-semantics; cf. Section 2). For this execution we used a randomized traversal strategy; that is, we prioritized the retrieval of Linked Data by using randomly chosen lookup priorities for all URIs that are discovered and need to be looked up during the execution process. By repeating this query execution five times, for each of these executions, we measured an overall execution time of 8.9 min (because all five executions eventually retrieve the same set of documents, which always requires almost the same amount of time). However, due to the random prioritization, the documents always arrive in a completely different order, which affects the time until all the data has been retrieved that is needed to compute any particular result element: In the best of the five cases, a first element of the result set can be returned after 9 sec, that is, 1.7% of the overall query execution time; on average however the five executions require 3.1 min (34.8%) to return a first result element, and the standard deviation of this average is as high as 1.3 min (14.6%).

The example illustrates that there exists a huge potential for optimizing the response times of traversal-based query executions (i.e., returning result elements as early as possible) and that these response times may vary significantly depending on the strategy chosen to traverse the queried Web of Linked Data. A desirable traversal strategy is one that prioritizes the lookup of URIs such that it discovers as early as possible the result-relevant documents (whose data can be used to compute at least one of the elements of the query result). Then, as soon as these documents arrive, a pipelined result construction process can compute and output result elements. The primary challenge in this context is that the URIs to be looked up are discovered only recursively, and we cannot assume up-to-date a priori information about what URIs will be discovered and which of the discovered URIs allow us to retrieve documents that are result-relevant. Given these issues, an investigation of possible approaches to prioritize URI lookups and their impact on the response times is an open research problem that is important for improving the user experience of applications that can be built on the traversal-based paradigm.

In this paper, we focus on this problem. To this end, we identify a diverse set of *14 different approaches to prioritize URI lookups* during a traversal-based query execution. None of these approaches assumes any a priori information about the queried Web. Then, as our main contribution, we conduct an experimental analysis to study the effects that each of these prioritization approaches can have on the response times of traversal-based query executions. This analysis is based on a comprehensive set of structurally diverse test Webs. We show that some of the approaches can achieve significant improvements over a breadth-first search baseline approach that looks up URIs on a first-come, first-served basis. However, we also observe that, even for the most promising ones of our approaches, there is a non-negligible number of cases in which

they perform worse than the baseline. Before describing the approaches (Section 4) and discussing our experiments in more detail (Sections 5–6), we briefly review the state of the art in querying Linked Data on the Web and elaborate more on the focus of our work.

2 Linked Data Query Processing

The prevalent query language used in existing work on querying Linked Data on the WWW is the basic fragment of SPARQL. Approaches to evaluate such basic graph patterns (BGPs) over Linked Data can be classified into traversal-based, index-based, and hybrid [7,11]. All these approaches compute a query result based on Linked Data that they retrieve *by looking up URIs during the query execution process*. Their strategy to select these URIs is where the approaches differ.

Traversal-based approaches perform a recursive URI lookup process during which they incrementally discover further URIs that can be selected for lookup. Existing work in this context focuses on techniques to implement such a traversal-based query execution [8,12,13]; additionally, as a well-defined foundation for these approaches, we have proposed a family of *reachability-based query semantics* for SPARQL that restrict the scope of any query to a query-specific *reachable subweb* [6]. To this end, the specification of any query in this context includes a set of seed URIs (in addition to the query pattern). Then, a document in the queried Web of Linked Data is defined to be *reachable* (and, thus, part of the reachable subweb) if it can be retrieved by looking up either a seed URI—in which case we call it a *seed document*—or a URI u such that (i) u occurs in an RDF triple of some other reachable document and (ii) u meets a particular *reachability condition* specified by the given reachability-based query semantics. For instance, such a condition may require that the triple in which the URI is found is a matching triple for any of the triple patterns in the given query. Our earlier work formalizes this condition in a reachability-based query semantics that we call c_{Match}*-semantics* [6].

Index-based approaches use a pre-populated index whose entries are URIs that can be looked up to retrieve Linked Data. Then, for any given query, such an approach uses its index to select a set of URIs whose lookup will result in retrieving query-relevant data. By relying on their index, index-based approaches fail to exploit query-relevant data added to indexed documents after building the index, and they are unaware of new documents. Existing work on such approaches focuses on different ways to construct the corresponding index [17], on techniques to leverage such an index [17], and on ranking functions that prioritize the lookup of the selected URIs in order to reduce response times [11,17]. The latter aims to achieve the same objective as our work in this paper. However, the ranking functions proposed for index-based approaches rely on statistical metadata that has been added to the index. For our work on traversal-based query executions we do not assume an a priori availability of any metadata whatsoever.

The only ***hybrid approach*** that has been proposed in the literature so far exploits an index to populate a prioritized list of seed URIs; additional URIs discovered during a subsequent traversal-based execution are then integrated into the list [11]. To this end, discovered URIs that are in the index (but have not been selected initially) are prioritized based on a ranking function that uses information from the index. For any URI for which no index entry exists, the approach simply uses as priority the number of retrieved Linked Data documents that mention the URI in some of their RDF triples (i.e., the number of known incoming links). One of the prioritization approaches that we analyze in this paper resembles the latter strategy (cf. Section 4.1).

3 Focus of Our Work

As discussed in the previous section, the prioritization of URI lookups is an idea that has been shown to be suitable to improve the response time of queries over the Web of Linked Data. However, the only systematic analyses of approaches that implement this idea focus on index-based query executions [11,17]. The approaches proposed in this context cannot be used for a traversal-based execution because they rely on statistical metadata that may be recorded when building an index but that is not a priori available to a (pure) traversal-based query execution system (which also rules out these approaches for non-indexed URIs in a hybrid system). Therefore, the overall goal of the work presented in this paper is to investigate URI prioritization approaches that can be used to reduce the response times of traversal-based query executions.

For this work we make minimal assumptions about how traversal-based query execution is implemented, which ensures independence of the peculiarities of any particular implementation techniques (such as those proposed in earlier work [8,12,13]). That is, we only make the general assumption that traversal-based query engines consist of a data retrieval component (DR-component) and a result construction component (RC-component), and these two components operate in parallel to execute a query as follows.

The DR-component receives Linked Data by looking up URIs. To this end, the component is equipped with a *lookup queue* that is initialized with the seed URIs of the given query. The component may use multiple *URI lookup threads*. Whenever such a thread is free, it obtains the next URI from the queue, looks up this URI on the Web, and scans the RDF triples that are contained in the document retrieved by the lookup. This scan has two goals: First, the triples may contain new URIs that can be scheduled for lookup. However, the lookup threads do not necessarily have to add all new URIs to the lookup queue. Instead, the DR-component can support an arbitrary reachability-based query semantics. That is, the component may schedule only those URIs for lookup that satisfy the reachability condition specified by the semantics. By doing so, the lookup threads incrementally discover (and retrieve) the specific reachable subweb that the given query semantics defines as the scope of the query. Hence, all triples scanned by the lookup threads—and only these—have to be considered to compute the sound and complete query result. Consequently, the second goal of scanning these triples is to identify triples that match a triple pattern in the given query. Any such matching triple is then sent to the RC-component, which starts processing them as soon as they arrive.

Regarding the RC-component we make only three assumptions: (i) it uses the incoming matching triples to compute the final query result, (ii) it processes intermediate results in a push-based manner, and (iii) as soon as an element of the final query result is ready, it is sent to the output. For techniques to implement such a push-based RC-component we refer to the literature [12,13] and to the extended version of this paper [10].

The whole query execution process continues until the DR-component has accessed all data from the query-specific reachable subweb and the RC-component has finished processing the resulting intermediate solutions. If the queried Web is distributed over a comparably slow network such as the Internet (as we assume in this paper), it is not surprising to observe that data retrieval is the dominating factor for query execution times. In fact, as we verify experimentally in the extended version of this paper, due to this dominance of data retrieval, the execution times of traversal-based query executions *over the WWW* are not affected at all by the order in which URIs are looked up [10]. For the same reason, however, the URI lookup order has a crucial impact on

the times required to find a specific number of result elements (as demonstrated by Example 1). Therefore, when aiming to minimize such response times, a suitable approach to prioritize URI lookups is of critical importance. We study 14 candidates in this paper.

In this study we focus on conjunctive queries (represented by BGPs) under the bag version of the aforementioned c_{Match}-semantics. This semantics is the most prominent reachability-based query semantics supported by the traversal-based approaches studied in the literature [8,12,13]. While, in theory, there exist an infinite number of other reachability-based query semantics and our experiments can be repeated for any of them, we conjecture that the results will be similar to ours because none of the approaches studied in this paper makes use of anything specific to c_{Match}-semantics. Using the bag version of c_{Match}-semantics allows us to focus on a notion of the response time optimization problem that is isolated from the additional challenge of avoiding the discovery of duplicates (which is an additional aspect of response time optimization under set semantics and worth studying as an extension of our work).

As a final caveat before going into the details, we emphasize the following limitations of our study: We ignore factors that may impact the response times of traversal-based queries but that cannot be controlled by a system that executes such queries (e.g., varying latencies when accessing different Web servers). Moreover, we focus only on approaches that do *not* assume any a priori information about the queried Web of Linked Data. That is, the topology of the Web or statistics about the data therein is unknown at the beginning of any query execution. This focus also excludes approaches that aim to leverage such information collected during earlier query executions (of course, studying such approaches is an interesting direction for future work). Similarly, we ignore the possibility to cache documents for subsequent query executions. While caching can reduce the time to execute subsequent queries [5], this reduction comes at the cost of potentially outdated results. However, studying approaches to balance the performance vs. freshness trade-off in this context is another interesting direction for future work.

4 Approaches to Prioritize URI Lookups

A variety of approaches to prioritize URI lookups are possible. In this section, we identify different classes of such approaches. Figure 1 illustrates our taxonomy. All these approaches assume that the lookup queue of the DR-component is maintained as a priority queue. Priorities are denoted by numbers; the greater the number, the higher the priority. URIs that are queued with the same priority are handled in a first-come, first-served manner (after all higher priority URIs have been looked up).

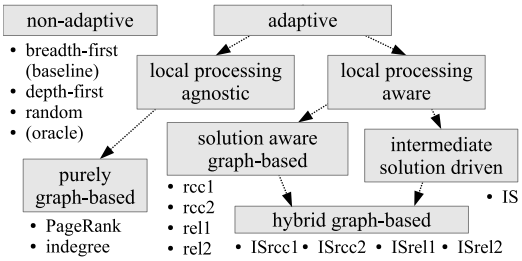


Figure 1. Approaches to prioritize URI lookups.

A first class includes *non-adaptive approaches* that determine a *fixed* priority for each URI when the URI is added to the lookup queue. A trivial example is to treat all URIs equal, which resembles a breadth-first traversal. We consider this approach as our baseline. In the extended version of this paper we also discuss depth-first and random as alternative non-adaptive approaches (Example 1 uses the latter). These turn out to be unsuitable for reducing the response times of traversal-based query executions [10].

4.1 Purely Graph-Based Approaches

In contrast to non-adaptive approaches, *adaptive approaches* may reprioritize queued URIs. A first class of such approaches is based on the idea of applying a vertex scoring method to a directed graph that represents the topology of the queried Web as discovered during the data retrieval process. Each vertex in this graph corresponds either to a retrieved document or to a queued URI. Each directed edge between two document vertices represents a data link that is established by URIs that occur in some RDF triple in the source document and that turned out to resolve to the target document when looked up. Directed edges from a document vertex to a URI vertex represent data links to documents that are yet to be retrieved. Obviously, such a graph is an incomplete model of the topology of the queried Web. However, as a side-effect of the data retrieval process, the DR-component obtains increasingly more information about the topology and, thus, can augment its model continuously. That is, any URI vertex becomes a document vertex after the corresponding URI has been looked up. If such a lookup results in discovering new URIs for the lookup queue, new URI vertices and connecting edges can be added to the graph. Similarly, new edges can be added if a retrieved document mentions URIs that either are already queued for lookup or have already been looked up.

Given such a graph, it is possible to apply a vertex scoring method and use the score of each URI vertex as the priority of the corresponding URI in the lookup queue. Whenever the DR-component extends the graph after completing some URI lookup, the vertex scores can be recomputed, and the priorities can be adapted accordingly.

A multitude of different vertex scoring methods exist. We select PageRank and in-degree-based scoring as two examples for our study. PageRank is a well-known method that uses an iterative algorithm to determine a notion of importance of vertices [14]. In-degree-based scoring is a less complex method that simply uses the number of incoming edges as the score of a vertex. Hereafter, we refer to the two resulting URI prioritization approaches as PageRank and indegree, respectively. We note that the latter approach is equivalent to the only existing proposal to prioritize URI lookups during traversal-based query executions [11]. However, its effectiveness has not been studied so far.

4.2 Solution-Aware Graph-Based Approaches

We now turn to *local processing aware approaches* that aim to leverage runtime information about the result construction process in the RC-component. To enable an implementation of these approaches, the traversal-based query execution engine must be extended with a *feedback channel* from the RC-component to the DR-component. Then, specific information required to prioritize URI lookups can be sent over this channel.

Given the possibility to obtain runtime information from the RC-component, we now can define graph-based URI prioritization approaches for which we use vertex scoring methods that leverage such runtime information. In this paper we focus on methods that are based on the number of solutions that retrieved documents have contributed to.

To enable the application of such methods, intermediate solutions must be augmented with provenance annotations. In particular, each intermediate solution must be annotated with a set of all documents that contributed a matching triple to the construction of that intermediate solution. To this end, before sending a matching triple to the RC-component, the DR-component augments this triple with metadata that identifies the source document of the triple. This document becomes the provenance of the initial intermediate solution that the RC-component generates from the matching triple. When

two intermediate solutions are joined in the RC-component, the union of their provenance annotations becomes the provenance of the resulting intermediate solution. Then, whenever an intermediate solution has been completed into a solution that is ready to be sent to the output, the RC-component uses the feedback channel to send the provenance annotation of this solution to the DR-component. The DR-component uses these annotations to maintain a *result contribution counter* (*RCC*) for every document vertex in the Web graph model that the component builds incrementally as described in Section 4.1. This counter represents the number of solutions that the document represented by the vertex has contributed to so far, which may increase as the query execution progresses.

Given these counters, we define four vertex scoring functions that can be applied to the Web graph model. Informally, for each vertex $v \in V$ in such a graph $G = (V, E)$, the *rcc–1 score* of v, denoted by $\mathrm{rccScore}_1(v)$, is the sum of the (current) RCCs of all document vertices in the in-neighborhood of v; and the *rel–1 score* of v, denoted by $\mathrm{relScore}_1(v)$, is the number of document vertices in the in-neighborhood of v whose RCC is greater than 1. Similarly, the *rcc–2 score* and *rel–2 score* of v, denoted by $\mathrm{rccScore}_2(v)$ and $\mathrm{relScore}_2(v)$, respectively, focus on the 2-step in-neighborhood. To define these scores formally, let $\mathrm{in}_k(v)$ denote the set of vertices in the k-step in-neighborhood of v, and, if v is a document vertex, let $\mathrm{rcc}(v)$ be its (current) RCC. Then, for each vertex $v \in V$ and $k \in \{1, 2\}$, the scoring functions are defined as follows:

$$\mathrm{rccScore}_k(v) = \textstyle\sum_{v' \in \mathrm{in}_k(v)} \mathrm{rcc}(v'), \quad \mathrm{relScore}_k(v) = \big|\{v' \in \mathrm{in}_k(v) \,|\, \mathrm{rcc}(v') > 0\}\big|.$$

These vertex scoring functions can be used by a graph-based approach to prioritize URI lookups (in the same manner as the PageRank and indegree approaches use the PageRank algorithm and indegree-based scoring, respectively). Hereafter, we refer to the four resulting URI prioritization approaches as rcc1, rcc2, rel1, and rel2, respectively.

4.3 Intermediate Solution Driven Approaches

An alternative class of local processing aware approaches use the aforementioned feedback channel to obtain information about all the intermediate solution mappings sent between operators in the RC-component. We focus on one such approach, denoted by *IS*, that assigns an initial priority of 0 to any new URI added to the lookup queue, and it reprioritizes queued URIs based on the following two assumptions:

A1: The greater the number of operators that have already processed a given solution mapping μ, the more likely it is that this intermediate solution μ can be completed into a solution μ' that covers the whole query and, thus, can be sent to the output.

A2: The documents that can be retrieved by looking up the URIs mentioned in a given (intermediate) solution mapping μ are the documents that are most likely to contain matching triples needed for completing μ into a solution.

Recall that the objective is to return solutions as early as possible. Hence, by assumption A2, it seems reasonable to increase the priority of a URI in the lookup queue if the URI is mentioned in an intermediate solution. Furthermore, by assumption A1, such an increase should be proportional to the number of operators that have already processed the intermediate solution. Then, to implement the IS approach, intermediate solutions do not only have to be sent between operators in the RC-component, but they also have to be sent over the feedback channel to the DR-component—after annotating them with the number of operators that contributed to their construction. Given an intermediate solution mapping μ with such a number, say $opcnt$, the IS approach iterates over all

variables that are bound by μ. For each such variable $?v \in \text{vars}(\mu)$, if μ binds the variable to a URI (i.e., $\mu(?v)$ is a URI) and this URI is queued for lookup with a priority value that is smaller than $opcnt$, then IS increases the priority of this URI to $opcnt$.

4.4 Hybrid Local Processing Aware Approaches

The idea of the IS approach (cf. Section 4.3) can be combined with the solution-aware graph-based approaches (cf. Section 4.2). To this end, the DR-component has to obtain via the feedback channel both the provenance annotation of each solution and all intermediate solution mappings. Based on the former, the DR-component increases the RCCs of document vertices in the Web graph model (as described in Section 4.2). The intermediate solutions are used to maintain an additional number for every URI that is queued for lookup; this number represents the maximum of the $opcnt$ values of all the intermediate solutions that bind some variable to the URI. Hence, initially (i.e., when the URI is added to the lookup queue) this number is 0, and it may increase as the DR-component gets to see more and more intermediate solutions via the feedback channel. Observe that this number is always equal to the lookup priority that the IS approach would ascribe to the URI. Therefore, we call this number the *IS-score* of the URI.

Given such IS-scores, we consider four different approaches to prioritize URI lookups, each of which uses one of the RCC-based vertex scoring functions introduced in Section 4.2. We call these approaches isrcc1, isrcc2, isrel1, and isrel2 (the name indicates the vertex scoring function used). Each of them determines the priority of a queued URI by multiplying the current IS-score of the URI by the current vertex score that their vertex scoring function returns for the corresponding URI vertex. Whenever the DR-component increases the IS-score of a URI, or the vertex score of the corresponding URI vertex changes, then the lookup priority of that URI is adapted accordingly.

4.5 Oracle Approach

We also want to gain an understanding of what response times a traversal-based query execution system could achieve if it had complete information of the queried Web (which is impossible in practice). To this end, we developed another approach assuming an *oracle* that, for each reachable document, knows (i) the URIs leading to the document and (ii) the final RCC of the document (i.e., the number of solutions of the complete query result that are based on matching triples from the document). Then, this oracle approach uses as priority of a URI lookup the final RCC of the document that will be retrieved by this lookup. As a consequence, retrieving documents with a greater final RCC has a higher priority. Clearly, without a priori information about the queried Web, a traversal-based system can determine such final RCCs only after retrieving all reachable documents—which is when it is too late to start prioritizing URI lookups. Hence, the oracle approach cannot be used in practice. However, for our experiments we performed a baseline-based "dry run" of our test queries and collected the information necessary to determine the RCCs that are required to execute the queries using the oracle approach.

5 Experimental Setup

In this section we specify the setup of our experiments. Although the execution of queries over Linked Data on the WWW is the main use case for the concepts in this paper, the WWW is not a controlled environment to run experiments on. For this reason, we set up a simulation environment consisting of two identical machines, each

with an Athlon 64 X2 dual core CPU, and 3.6 GB of main memory. Both machines use an Ubuntu 12.04 LTS operating system with Sun Java 1.6.0 and are connected via a fast university network. One machine runs a Tomcat server (7.0.26) with a Java servlet that can simulate different Webs of Linked Data (one at a time); the documents of these Webs are materialized on the machine's hard disk. The other machine executes queries over such a simulated Web by using an in-memory, Java implementation of a traversal-based query engine. To rule out any effects of parallelized URI lookups as a factor that may influence our measurements we set up the system to use a single lookup thread.

In the following, we specify the Webs of Linked Data simulated for our experiments, the corresponding test queries, and the metrics that we use. Software and data required for our experiments are available online at http://squin.org/experiments/ISWC2016/.

5.1 Test Webs

The goal of our experiments is to investigate how the different URI prioritization approaches impact the response times of traversal-based query executions. This impact (as well as the chance to observe it) may be highly dependent on how the queried Web of Linked Data is structured and how data is distributed. Therefore, we generated multiple test Webs for our experiments. To be able to meaningfully compare measurements across our test Webs, we used the same base dataset for generating these Webs.

We selected as base dataset the set of RDF triples that the data generator of the Berlin SPARQL Benchmark (BSBM) suite [1] produces for a scaling factor of 200. This dataset, hereafter denoted by G_{base}, consists of 75,150 RDF triples and describes 7,329 entities, each of which is identified by a unique URI. Let U_{base} denote the set consisting of these 7,329 URIs. Hence, the subject of any triple $\langle s, p, o\rangle \in G_{\mathsf{base}}$ is such a URI (i.e., $s \in U_{\mathsf{base}}$), and the object o either is a literal or also a URI in U_{base}.

Every test Web that we generated from this base dataset consists of 7,329 documents, each of which is associated with a different URI in U_{base}. To distribute the triples of G_{base} over these documents, we partitioned G_{base} into 7,329 potentially overlapping subsets (one for each document). First, we always placed any base dataset triple whose object is a literal into the subset of the document for the subject of that triple. Next, for any of the other base dataset triples $\langle s, p, o\rangle \in G_{\mathsf{base}}$ (whose object o is a URI in U_{base}), we considered three options: placing the triple (i) into both the documents for s and for o—which establishes a bidirectional data link between both documents, (ii) into the document for s only—which establishes a data link from that document to the document for o, or (iii) into the document for o only—which establishes a data link to the document for s. It is easy to see that choosing among these three options impacts the link structure of the resulting test Web (note that the choice may differ for each triple).

We exploited this property to systematically generate test Webs with different link structures. That is, we applied a random-based approach that, for every generated test Web, uses a particular pair of probabilities (ϕ_1, ϕ_2) as follows: For every base dataset triple $\langle s, p, o\rangle \in G_{\mathsf{base}}$ with $o \in U_{\mathsf{base}}$, we chose the first option with a probability of ϕ_1; otherwise, ϕ_2 is the (conditional) probability of choosing the second option over the third. To cover the whole space of possible link structures in a systematic manner, we have used each of the twelve pairs $(\phi_1, \phi_2) \in \{0, 0.33, 0.66\} \times \{0, 0.33, 0.66, 1\}$ to generate twelve test Webs $W^{0,0}_{\mathsf{test}}, \ldots, W^{66,100}_{\mathsf{test}}$, and we complemented them with the test Web W^{100}_{test} that we generated using probability $\phi_1 = 1$ (in which case ϕ_2 is irrelevant).

While these 13 test Webs cover a wide range of possible link structures, we are also interested in an additional test Web whose link structure is most representative of real

Linked Data on the WWW. To identify a corresponding pair of probabilities (ϕ_1, ϕ_2) we analyzed the 2011 Billion Triple Challenge dataset [3]. For this corpus of real Linked Data we identified a ϕ_1 of 0.62 and a ϕ_2 of 0.47. Given this pair of probabilities, we used our base dataset to generate another test Web, $W_{\mathsf{test}}^{62,47}$. In this paper we discuss primarily the measurements obtained by querying this test Web. However, for our analysis we also queried the other test Webs; the measurements of all these query executions contribute to our empirical comparison of the URI prioritization approaches (cf. Section 6.6).

We emphasize that a systematic creation of test Webs with different link structures as achieved by the given, random-based approach requires a base dataset that has a high degree of structuredness, which is the case for our BSBM dataset [2]. On the other hand, even if the base dataset is highly structured, our random-based approach ensures that the documents in each generated test Web (except W_{test}^{100}) contain data with varying degrees of structuredness, which reflects most of the Linked Data on the WWW [2].

5.2 Test Queries

For our experiments we use six SPARQL basic graph patterns (BGPs) under c_{Match}-bag-semantics (cf. Section 2); as seed URIs, we use all URIs in the given BGP, respectively. These queries, denoted by Q1 to Q6, are listed in the extended version of this paper [10].

We created these six queries so that they satisfy the following three requirements: First, each of these queries can be executed over all our test Webs. Second, the queries differ w.r.t. their syntactical structure (shape, size, etc.). Third, to avoid favoring any particular traversal strategy, the reachable subwebs induced by the queries differ along various dimensions. For instance, Table 1 lists several properties of the six query-specific reachable subwebs of test Web $W_{\mathsf{test}}^{62,47}$. These properties are the number of reachable documents, the number of edges between these documents in the link graph of the reachable subweb, the number of strongly connected components and the diameter of the link graph, the number of reachable documents that are *result-relevant* (i.e., their data is required for at least one solution of the corresponding query result), the percentage of reachable documents that are result-relevant, the mean lengths of the shortest paths (in the link graph) from seed documents to these relevant documents, the lengths of the shortest and the longest of these shortest paths, and similar statistics for the reachable documents that are not result-relevant. Additionally, Table 1 lists the cardinality of the corresponding query results. We emphasize that a computation of any of the properties in Table 1 requires information that a traversal-based system discovers only during query execution. Hence, such statistics can be computed only after *completing* a traversal-based query execution and, thus, they cannot be used for query optimization.

By comparing the properties in Table 1, it can be observed that our six test queries induce a very diverse set of reachable subwebs of test Web $W_{\mathsf{test}}^{62,47}$. In an earlier, more detailed analysis of these queries we make the same observation for the other 13 test Webs [9]. Moreover, if we consider each query in separation and compare its reachable

Query	link graph of reachable subweb				result-relevant reachable documents					res.-irrel. reach. documents			result
	# of docs	# of edges	str. conn. components	dia-meter	# of docs	% of all reach.docs	shortest paths from seeds mean (st.dev)	min	max	shortest paths from seeds mean (st.dev)	min	max	cardi-nality
Q1	3818	10007	413	8	572	15.0%	1.12 (±0.43)	1	3	1.69 (±0.93)	1	3	2481
Q2	214	627	8	15	22	10.3%	2.34 (±1.70)	1	8	5.04 (±1.40)	2	8	34
Q3	234	410	57	6	3	1.3%	1.41 (±0.50)	1	2	2.74 (±0.53)	1	3	4
Q4	1098	7805	36	12	43	3.9%	1.38 (±0.73)	1	3	3.49 (±0.98)	1	5	804
Q5	333	2340	14	10	36	10.8%	2.21 (±0.78)	1	4	3.83 (±0.37)	3	5	116
Q6	2232	6417	88	45	12	0.5%	2.40 (±0.78)	1	4	4.08 (±1.34)	1	8	28

Table 1. Statistics about the reachable subwebs of test queries Q1–Q6 over test Web $W_{\mathsf{test}}^{62,47}$.

subwebs across the different test Webs, we observe a similarly high diversity [9]. Hence, these six queries in combination with all 14 test Webs represent a broad spectrum of test cases. That is, we have some test cases that reflect interlinkage characteristics of a real snapshot of Linked Data on the WWW (i.e., $W_{\mathsf{test}}^{62,47}$) and others that systematically cover other possible interlinkage characteristics ($W_{\mathsf{test}}^{0,0}$ ... W_{test}^{100}).

5.3 Metrics

For each solution that our test system computes during a query execution, it measures and records the fraction of the overall execution time after which the solution becomes available. An example of such numbers for the first reported solution are the percentages given in Example 1. For our analysis we focus primarily on the two extreme cases: the relative response times for a first solution and the relative response times for the last solution. The former is interesting because it identifies the time after which users can start looking over some output for their query; the latter marks the availability of the complete result (even if the system cannot verify the completeness at this point). Hence, we define two metrics. Let *exec* be a query execution; let t_{start}, t_{end}, t_{1st}, and t_{last} be the points in time when *exec* starts, ends, returns a first solution, and returns the last solution, respectively. The *relative first-solution response time* (*relRT1st*) and the *relative complete-result response time* (*relRTCmpl*) of *exec* are defined as follows:

$$\mathsf{relRT1st} = \frac{t_{\mathsf{1st}} - t_{\mathsf{start}}}{t_{\mathsf{end}} - t_{\mathsf{start}}} \quad \text{and} \quad \mathsf{relRTCmpl} = \frac{t_{\mathsf{last}} - t_{\mathsf{start}}}{t_{\mathsf{end}} - t_{\mathsf{start}}}.$$

We can use such relative metrics for our study because, for each query, the overall query execution time is always the same, independent of the URI prioritization approach (cf. Section 3). The advantage of relative metrics is that they directly show the differences in response times that can be achieved by different URI prioritization approaches *relative to each other*. Measuring absolute times—such as the times that Example 1 provides in addition to the percentages—would not provide any additional insight for such a comparison. Moreover, absolute times that we may measure in our simulation environment are mostly a function of how fast our simulation server responds to URI requests. Hence, such absolute times in our simulation would be quite different from what could be measured for queries over the "real" Web of Linked Data (such as in Example 1).

To increase the confidence in our measurements we repeat every query execution five times and report the geometric mean of the measurements obtained by the five executions. The confidence intervals (i.e., error bars) in the charts in this paper represent one standard deviation. To avoid measuring artifacts of concurrent query executions we execute queries sequentially. To also exclude possible interference between subsequent query executions we stop and restart the system between any two executions.

6 Experimental Results

To experimentally analyze the URI prioritization approaches introduced in Section 4 we used each of these approaches for traversal-based query executions over our test Webs. The charts in Figure 2 illustrate the mean relRT1st and the mean relRTCmpl measured for the query executions over test Web $W_{\mathsf{test}}^{62,47}$ (in some cases the bars for relRT1st are too small to be seen). For instance, the leftmost bars in Figure 2(a) indicate that the baseline executions of query Q1 returned a first solution of the query result after 26.5% of the overall query execution time, and it took them about 99% of the time to complete the query result. In this section, we discuss these measurements, as well as further

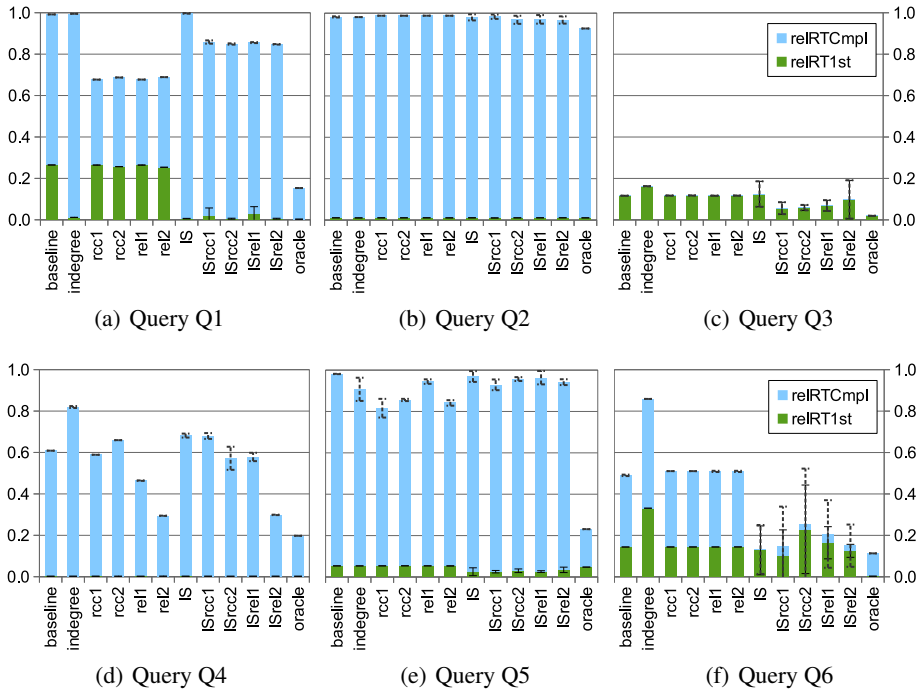
noteworthy behavior as observed for query executions over the other test Webs. The discussion is organized based on the classification of URI prioritization approaches as introduced in Section 4 (Figure 1). However, we begin with some general observations.

6.1 General Observations

A first, unsurprising observation is that, in almost all cases, none of the approaches achieves response times that are smaller than the response times achieved by the oracle approach. However, we also notice a few (minor) exceptions. These exceptions can be explained by the fact that—independent of what URI prioritization approach is applied—the DR-component discovers the URIs to be looked up only gradually. Then, by *greedily* ordering the currently available URIs (based on our pre-computed RCCs), the oracle approach may only achieve a local optimum but not the global one.

Ignoring the oracle approach for a moment, we note that approaches that achieve a good relRT1st for a query do not necessarily also achieve a good relRTCmpl for that query.

Another general observation is that, by using different URI prioritization approaches to execute the same query over the same test Web, the number of intermediate solutions processed by our system can vary significantly, and so does the number of priority changes initiated by the adaptive approaches. These variances indicate that the amount of computation within our system can differ considerably depending on which URI prioritization approach is used. Nonetheless, in all our experiments the overall time to execute the same query over the same test Web is always almost identical for the different approaches! This fact again illustrates the dominance of the data retrieval fraction



(a) Query Q1 (b) Query Q2 (c) Query Q3

(d) Query Q4 (e) Query Q5 (f) Query Q6

Figure 2. Relative response times for queries Q1 to Q6 over test Web $W_{\text{test}}^{62,47}$ as achieved by employing the different approaches to prioritize URI lookups.

of query execution time (cf. Section 3) and, thus, is a strong verification of the comparability of our relative measurements. The only exception is the PageRank approach for which query execution times range from 120% to 320% of the execution times measured for the other approaches. Hence, in contrast to the additional computation required for each of the other approaches, the frequent execution of the iterative PageRank algorithm becomes a non-negligible overhead. As a result, the PageRank approach cannot compete with the other approaches and, thus, we ignore it in the remainder.

Finally, in Figure 2(c) (for query Q3 over test Web $W_{\mathsf{test}}^{62,47}$), we note that, for all approaches, the differences between the time needed to return a first solution and the time to return the last solution are insignificant. We explain this phenomenon as follows: Only three of the 234 reachable documents for Q3 over $W_{\mathsf{test}}^{62,47}$ contribute to the query result and this result consists of four solutions (cf. Table 1). It turns out that the computation of each of these four solutions requires data from each of the three result-relevant documents. Hence, only after (and as soon as) the last of these three documents has been retrieved, the system can compute and return all four solutions.

6.2 Evaluation of the Purely Graph-Based Approaches

After ruling out the PageRank approach (cf. Section 6.1), indegree is the only remaining purely graph-based approach in our experiments. We observe that this approach is often worse and only in a few cases better than the baseline approach (for both relRT1st and relRTCmpl). The reason for the negative performance of this approach—as well as any other possible purely graph-based approach—is that the applied vertex scoring method rates document and URI vertices only based on graph-specific properties, whereas the result-relevance of reachable documents is independent of such properties. In fact, in our earlier work we show empirically that there does not exist a correlation between the result-relevance—or irrelevance—of reachable documents and the indegree of the corresponding document vertices in the Web graph model (similarly, for the PageRank, the HITS scores, the k-step Markov score, and the betweenness centrality) [9].

6.3 Evaluation of the Solution-Aware Graph-Based Approaches

In contrast to the purely graph-based approaches, the solution-aware graph-based approaches (rcc1, rcc2, rel1, and rel2) employ vertex scoring methods that make use of information about result-relevant documents as discovered during the query execution process. We notice that, until such information becomes available (that is, not before a first query solution has been computed), these methods rate all vertices equal. As a consequence, all URIs added to the lookup queue have the same priority and are processed in the order in which they are discovered. Hence, until a first solution has been computed, the solution-aware graph-based approaches behave like the baseline approach. Therefore, these approaches always achieve the same relRT1st as the baseline.

Once a first set of result-relevant documents can be identified, the solution-aware graph-based approaches begin leveraging this information. As a result, for several query executions in our experiments, these approaches achieve a relRTCmpl that is significantly lower than the baseline. Moreover, for the majority of query executions for which this is not the case, the relRTCmpl achieved by the solution-aware graph-based approaches is comparable to the baseline. In the following, we identify characteristics of reachable subwebs that are beneficial for our four solution-aware graph-based approaches (for a more detailed discussion refer to the extended version of this paper [10]).

A necessary (but not necessarily sufficient) characteristic is that every reachable document that is result-relevant must have at least one in-neighbor that is also result-relevant (for rel2 and rcc2 it may also be a 2-step in-neighbor). However, even if the in-neighborhood of a relevant document d contains some other relevant documents, the solution-aware graph-based approaches can increase the retrieval priority of document d only if the relevance of at least one of these other documents, say d', is discovered before the retrieval of d. This is possible only if the relevance of d' can be attributed to its contribution to some query solution whose computation does not require document d.

Hence, an early discovery of a few first solutions may increase chances that the solution-aware graph-based approaches retrieve all relevant documents early, which then leads to smaller complete-result response times (relRTCmpl). However, there are also cases in which the identification of relevant documents may mislead these approaches; in particular, if some relevant documents link to many irrelevant documents. A special case that is particularly worse for rcc1 and rcc2 is the existence of a result-relevant document d with an RCC that is significantly higher than the RCCs of the other relevant documents in the corresponding subweb; such a high RCC may dominate the RCC-based scores in the in-neighborhood of document d. The Q4-specific reachable subweb of test Web $W_{\mathsf{test}}^{62,47}$ is an example of such a case (cf. Figure 2(d)).

6.4 Evaluation of the Intermediate Solution Driven Approaches

Intermediate solution driven approaches (including the hybrid approaches analyzed in the next section) use information about all intermediate solutions sent between operators in the RC-component. Regarding these approaches, we notice a high variation in our measurements (observe the error bars in Figure 2). We attribute this variation to the multithreaded execution of all operators in the RC-component of our traversal-based query engine (which we describe in detail in the extended version of this paper [10]). Due to multithreading, the exact order in which intermediate solutions appear in the RC-component and are sent to the DR-component is nondeterministic. As a result, the intermediate solution driven adaptation of the priorities of URIs that are queued for lookup becomes nondeterministic. Then, due to this nondeterminism, the order in which reachable documents are retrieved may differ for repeated executions with the same prioritization approach. Such differences may cause different response times because the retrieval order of documents determines which intermediate solutions can be generated at which point during the query execution process.

Irrespective of the variations, our measurements indicate that, in a number of cases, the IS approach can achieve an advantage over the baseline approach. For instance, compare the relRT1st values in Figure 2(a) or the relRTCmpl values in Figure 2(f). However, there also exist a significant number of cases in which IS performs worse than the baseline approach (e.g., query Q4 over test Web $W_{\mathsf{test}}^{62,47}$; cf. Figure 2(d)).

6.5 Evaluation of the Hybrid Approaches

For the hybrid approaches (isrcc1, isrcc2, isrel1, isrel2) we first notice that they all achieve similar response times in many cases. More importantly, however, these response times are comparable, or at least close, to the best of either the response times achieved by the solution-aware graph-based approaches or the response times of the IS executions.

A typical example are the executions of Q1 over test Web $W_{\mathsf{test}}^{62,47}$ (cf. Figure 2(a)). On one hand, the hybrids achieve complete-result response times (relRTCmpl) for this query that are smaller than the baseline—which is also the case for the solution-aware

graph-based approaches but not for the IS-based executions. On the other hand, instead of also achieving first-solution response times (relRT1st) as achieved by the solution-aware graph-based approaches (which are as high as the baseline), the hybrids achieve a relRT1st that is as small as what the IS-based executions achieve. Regarding relRT1st we recall that the solution-aware graph-based approaches cannot be better than the baseline. The latter observation shows that this is not the case for the hybrid approaches. On the contrary, even if each hybrid approach is based on a solution-aware graph-based approach, their combination with intermediate solution driven functionality enables the hybrid approaches to outperform the baseline in terms of relRT1st.

6.6 Comparison

Our measurements show that there is no clear winner among the URI prioritization approaches studied in this paper. Instead, for each approach, there exist cases in which the approach is better than the baseline and cases in which the approach is worse.

Table 2 quantifies these cases; that is, the table lists the percentage of cases in which the response times achieved by each approach are at least 10% better (resp. 10% worse) than the baseline. For this comparison, we consider the executions of all six test queries *over all 14 test Webs* (i.e., 84 cases for each approach), and we use the threshold of 10% to focus only on noteworthy differences to the baseline. In addition to relRT1st and relRTCmpl, the table also covers *relative 50% response time* (*relRT50*); that is, the fraction of the overall execution time after which 50% of all solutions of the corresponding query result have been computed.

	relRT1st		relRT50		relRTCmpl	
approach	worse	better	worse	better	worse	better
DFS	23.2%	26.1%	58.9%	17.8%	53.6%	10.1%
random	13.0%	27.5%	58.9%	8.2%	59.4%	8.7%
indegree	21.7%	21.7%	65.8%	4.1%	50.7%	5.8%
rcc1	0.0%	0.0%	4.1%	1.4%	7.2%	24.6%
rcc2	0.0%	0.0%	2.7%	2.7%	4.1%	20.3%
rel1	0.0%	0.0%	5.5%	1.4%	11.6%	29.0%
rel2	0.0%	0.0%	11.0%	0.0%	2.9%	26.1%
IS	7.2%	31.9%	15.1%	27.4%	26.1%	10.1%
isrcc1	2.9%	30.4%	5.5%	26.0%	14.5%	18.8%
isrcc2	5.8%	33.3%	5.5%	24.7%	13.0%	26.1%
isrel1	0.0%	33.3%	2.7%	24.7%	15.9%	26.1%
isrel2	2.9%	31.9%	4.1%	23.3%	11.6%	26.1%
oracle	0.0%	35.3%	0.0%	41.2%	0.0%	64.7%

Table 2. Percentage of cases in which the approaches achieve response times that are at least 10% worse (resp. 10% better) than the baseline.

For both relRT1st and relRT50, we observe that isrel1 is the best of the approaches tested (ignoring the oracle approach which cannot be used in practice; cf. Section 4.5). Although the other intermediate solution driven approaches (IS, isrel2, isrcc1, isrcc2) have a similarly high number of cases in which they are at least 10% better than the baseline, these approaches have a higher number of cases in which they are at least 10% worse. We also notice that, as discussed in Section 6.3, for relRT1st, the solution-aware graph-based approaches (rcc1, rcc2, rel1, rel2) behave like the baseline.

For relRTCmpl, we observe some differences. The hybrid approaches (isrcc1, ..., isrel2) still have a comparably high number of cases in which they are at least 10% better than the baseline, but they also have a significant number of noteworthy cases in which they are worse. IS has an even higher number of such worse cases. In contrast, the solution-aware graph-based approaches are more suitable, with rel2 being the best choice.

In summary, to return some solutions of query results as early as possible, isrel1 appears to be the most suitable choice among the approaches studied in this paper. However, if the objective is to reduce complete-result response times (relRTCmpl), the solution-aware graph-based approaches are usually better suited; in particular, rel2. In the extended version of the paper we additionally show that, by and large, these general findings are independent of whether the queried Web is densely populated with bidirectional data links (i.e., $\phi_1 \geq 0.66$) or sparse (i.e., $\phi_1 \leq 0.33$) [10].

7 Conclusions

This is the first paper that studies the problem of optimizing the response times of traversal-based query executions over Linked Data. In particular, we focus on the fundamental problem of fetching result-relevant data as early as possible. To this end, we introduce heuristics-based approaches to prioritize URI lookups during data retrieval and analyze their impact on response times. For this experimental analysis we use a broad range of simulated, structurally diverse Webs of Linked Data. One of these test Webs reflects interlinkage characteristics of a real snapshot of Linked Data on the WWW, and the others systematically cover other possible interlinkage characteristics as may reflect other Webs of Linked Data (e.g., within the intranet of an enterprise).

Our experiments show that some of the approaches can achieve noteworthy improvements over the baseline in a significant number of cases. However, even for the best URI prioritization approaches in this paper, there exist cases in which the baseline approach achieves better response times. Moreover, a comparison to the oracle approach shows that there is further room for improvement. A promising direction of future work are approaches that collect statistics during (traversal-based) query executions and leverage these statistics to optimize the response times for subsequent queries.

References

1. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. Sem. Web and Inf. Sys. 5(2) (2009)
2. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. In: Proc. of ACM SIGMOD (2011)
3. Harth, A.: Billion Triples Challenge Data Set. http://km.aifb.kit.edu/projects/btc-2011 (2011)
4. Harth, A., Hose, K., Schenkel, R. (eds.): Linked Data Management. Chapman & Hall (2014)
5. Hartig, O.: How Caching Improves Efficiency and Result Completeness for Querying Linked Data. In: Proceedings of the 4th Linked Data on the Web Workshop (LDOW) (2011)
6. Hartig, O.: SPARQL for a Web of Linked Data: Semantics and Computability. In: Proceedings of the 9th Extended Semantic Web Conference (ESWC) (2012)
7. Hartig, O.: An Overview on Execution Strategies for Linked Data Queries. Datenbank-Spektrum 13(2), 89–99 (2013)
8. Hartig, O., Bizer, C., Freytag, J.C.: Executing SPARQL Queries over the Web of Linked Data. In: Proceedings of the 8th International Semantic Web Conference (ISWC) (2009)
9. Hartig, O., Özsu, M.T.: Reachable Subwebs for Traversal-Based Query Execution. In: Proceedings of the 23rd International World Wide Web Conference (WWW) (2014)
10. Hartig, O., Özsu, M.T.: Walking without a Map: Optimizing Response Times of Traversal-Based Linked Data Queries (Extended Version). CoRR abs/1607.01046 (2016)
11. Ladwig, G., Tran, T.: Linked Data Query Processing Strategies. In: Proc. of ISWC (2010)
12. Ladwig, G., Tran, T.: SIHJoin: Querying Remote and Local Linked Data. In: ESWC (2011)
13. Miranker, D.P., Depena, R.K., Jung, H., Sequeda, J.F., Reyna, C.: Diamond: A SPARQL Query Engine, for Linked Data Based on the Rete Match. In: Proc. of AImWB (2012)
14. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Tech. Rep. 1999-66, Stanford InfoLab (Nov 1999)
15. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In: Proc. of ISWC (2011)
16. Umbrich, J., Hogan, A., Polleres, A., Decker, S.: Link Traversal Querying for a Diverse Web of Data. Semantic Web Journal 6(6), 585–624 (2015)
17. Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres, A.: Comparing Data Summaries for Processing Live Queries over Linked Data. World Wide Web 14(5–6) (2011)