Demonstration Paper

Demonstration of Link Traversal SPARQL Query Processing over the Decentralized Solid Environment

Ruben Taelman Ruben Verborgh
IDLab, Department of Electronics and Information Systems, Ghent University – imec, Ghent, Belgium
ruben.taelman@ugent.be

ABSTRACT

To tackle economic and societal problems originating from the centralization of Knowledge Graphs on the Web, there has been an increasing interest towards *decentralizing* Knowledge Graphs across a large number of small authoritative sources. In order to effectively build user-facing applications, there is a need for efficient query engines that abstract the complexities around accessing such massively Decentralized Knowledge Graphs (DKGs). As such, we have designed and implemented novel Link Traversal Query Processing algorithms into the Comunica query engine framework that are capable of efficiently evaluating SPARQL queries across DKGs provided by the Solid decentralization initiative. In this article, we demonstrate this query engine through a Web-based interface over which SPARQL queries can be executed over simulated and real-world Solid environments. Our demonstration shows the benefits of a traversal-based approach towards querying DKGs, and uncovers opportunities for further optimizations in future work in terms of both query execution and discovery algorithms.

Video: https://www.youtube.com/watch?v=4WHWgWWZ_aQ

1 INTRODUCTION

Most research on the optimization and execution of SPARQL queries [1] over Knowledge Graphs [2] focuses on centralized use cases, in which queries are to be executed over a single dataset, in which queries can be optimized using traditional *optimize-then-execute* techniques [3, 4].

Due to issues in recent years involving personal data exploitation on the Web, there has been increasing interest in the decentralization of such personal data. This has materialized into various decentralization initiatives, such as Solid [5], Bluesky [6], and Mastodon [7]. What is common among these initiatives, is that they distribute personal and permissioned data across a large number of authoritative Web sources, which leads to the emergence of Decentralized Knowledge Graphs (DKGs). For this work, we limit our scope to RDF-based initiatives such as Solid, as the use of IRIs in RDF enable convenient integration of data across multiple sources. Solid enables users to host and control *personal data pods*, which are data sources into which hierarchies of RDF documents can be stored.

In order to *find* data across such massive distributions of data across the Web, there is a need for efficient query processing techniques. While techniques have been introduced that enable the execution of SPARQL federated queries [8, 9, 10], they are optimized for handling a *small* number (~10) of *large* sources [11], whereas DKGs such as Solid are characterized by a *large* number (>1000) of *small* sources. Additionally, federated SPARQL query processing assumes sources to be known prior to query execution, which is not feasible in DKGs due to the lack of a central index. Hence, these techniques are ill-suited for the envisaged scale of distribution in DKGs.

To cope with these problems, alternative techniques have been introduced recently. ESPRESSO [12] was introduced as an approach that builds distributed indexes for Solid pods which can be accumulated in a single location. This accumulated index can then be queried using keyword search to find relevant pods to a query, after which these relevant pods can be queried using federated SPARQL processing techniques. This approach depends on placing trust over personal data in this single accumulated indexer. PODQUERY [13] is another approach that involves placing a SPARQL query engine agent in front of a Solid pod, which enables full SPARQL queries to be executed over single Solid pods. This approach does not consider query execution across multiple pods. In recent work [14], we proposed making use of Link Traversal Query Processing (LTQP) for querying across one or more Solid pods. LTQP is derived from the idea of *SQL-based query execution over the Web* [15, 16] and the concept of *focused crawling* [17, 18]. LTQP starts query execution given a set of seed sources, from which links are recursively followed between RDF documents to discover additional data to consider during query execution, In this previous work, we introduced various LTQP techniques that understand the structural properties of Solid pods, and use this to optimize LTQP in terms of the number of links that need to be followed. Such a traversal-based approach does not rely on prior indexes over Solid pods, and can query over live data that is spread over multiple pods. Results have shown [14] that non-complex queries can be completed in the order of seconds, with first results showing up in less than a second. For more complex queries in terms of the number of triple patterns, results show that more fundamental optimization work is needed into adaptively optimizing the query plan upon newly discovered information and reducing the number of links to be followed.

The focus of this article is on demonstrating the implementation of a query engine that can execute SPARQL queries across Solid pods using the LTQP techniques introduced in [14]. This query engine is open-source, and is implemented in a modular way so it can serve future research in this domain. We demonstrate this query engine through a Web-based user interface, in which preconfigured or manually created SPARQL queries can be executed across both simulated or real-world Solid pods. In the next section, we explain our approach, after which we discuss our implementation in Section 3. In Section 4, we describe our Web-based demonstration interface, after which we conclude in Section 5.

2 APPROACH

Our previously introduced approach [14] enables query execution across one or more Solid pods without the need to build any prior indexes. For this, we build on top of the Link Traversal Query Processing (LTQP) paradigm, whereby the query engine maintains an internal *link queue* which is initialized through a set of *seed URLs*. This set of seed URLs determine the starting point for traversal, and may either be user-provided, or can be derived from the URLs mentioned in the given SPARQL query. The query engine will start the process by continuously iterating over this link queue, dereferencing each link, and adding all discovered RDF triples into an internal triple source that can continuously grow. Furthermore, for each dereferenced link with resulting RDF triples, the engine finds links to other sources, which are appended to the link queue. Different strategies exist for determining these links [19, 14]. During the processing of this link queue, the actual query processing happens in parallel over the continuously growing internal triple. This processing starts by building a logical query plan using the zero-knowledge query planning technique [20], which is necessary due to the fact that LTQP has no access to prior statistical information about the data to be queried over. After that, the plan is executed following an iterator-based pipeline approach [21], which considers the execution plan

as a pipeline [22] of iterator-based physical operators. This pipelined approach allows results to be returned to the end-user, even though the link traversal and query processing may still be running for a longer time. A visualization of the architecture of our approach can be seen in Fig. 1.

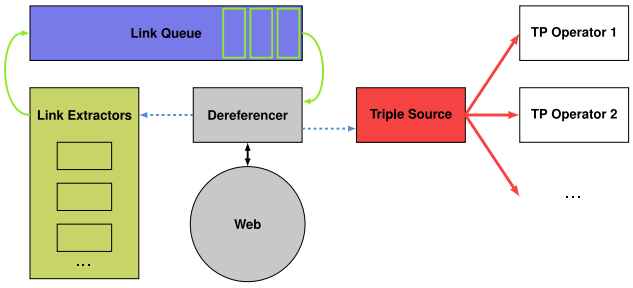

Fig. 1: Link queue, dereferencer, and link extractors feeding triples into a triple source, producing triples to tuple-producing operators in a pipelined query execution.

Instead of considering all possible links that are discovered in the RDF triples, we apply various Solid-specific [14] and Solid-agnostic [19] link extraction strategies. For example, all Solid pods make use of the Linked Data Platform (LDP) specification [23] to provide an overview of all RDF documents inside a pod, which may be nested in a hierarchy of *containers*. An example of such a container can be seen in Listing 1, which contains links to another document and two containers. In order to indicate the existence of such Solid pods, agents within the Solid ecosystem (typically persons of organizations) may link to their pods via their identifying *WebID* [24]. Through this WebID, agents can be uniquely identified via a URL. After dereferencing this URL, an RDF document can be returned that can contain a link to the user's Solid pod and other basic information such as name and contact details. An example of such a WebID document is shown in Listing 2. Furthermore, Solid pods can expose a Type Index [25], which contains a list of RDF classes for which instances exist in this pod, together with links to RDF documents containing such instances. An example of such a Type Index can be seen in Listing 3, which contains entries for posts and comments.

```
PREFIX ldp: <http://www.w3.org/ns/ldp#>
<> a ldp:Container, ldp:BasicContainer, ldp:Resource;
   ldp:contains <file.ttl>, <posts/>, <profile/>.
<file.ttl> a ldp:Resource.
<posts/> a ldp:Container, ldp:BasicContainer, ldp:Resource.
<profile/> a ldp:Container, ldp:BasicContainer, ldp:Resource.
```

Listing 1: An LDP container in a Solid data vault containing one file and two directories in the RDF Turtle serialization.

```
PREFIX pim: <http://www.w3.org/ns/pim/space#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX solid: <http://www.w3.org/ns/solid/terms#>
<#me> foaf:name "Zulma";
      pim:storage </>;
      solid:oidcIssuer <https://solidcommunity.net/>;
      solid:publicTypeIndex </publicTypeIndex.ttl>.
```

Listing 2: A simplified WebID profile in Turtle.

```
PREFIX ldp: <http://www.w3.org/ns/ldp#>
<> a solid:TypeIndex ;
   a solid:ListedDocument.
<#ab09fd> a solid:TypeRegistration;
  solid:forClass <http://example.org/Post>;
  solid:instance </public/posts.ttl>.
<#bq1r5e> a solid:TypeRegistration;
  solid:forClass <http://example.org/Comment>;
  solid:instanceContainer </public/comments/>.
```

Listing 3: Example of a type index with entries for posts and comments in RDF Turtle.

3 IMPLEMENTATION

Our approach has been implemented using the JavaScript/TypeScript-based Comunica SPARQL framework [26]. Following the modular architecture of Comunica, we have implemented our approach as several small modules, which allows modules to be enabled or disabled using a plug-and-play configuration system for the flexible combination of techniques during experimentation. This implementation has full support for SPARQL 1.1 queries, which consists of pipelined implementations of all monotonic SPARQL operators. As such, our system is able to execute SPARQL queries over Solid pods using a traversal-based approach. Since certain documents within Solid pods may exist behind document-level access control, our implementation supports authentication. This allows users to log into the query engine using their Solid WebID, after which the query engine will execute query on their behalf across all data the user can access.

Our implementation is available under the MIT license at https://github.com/comunica/comunica-feature-link-traversal, and via the npm package manager as `@comunica/query-sparql-link-traversal-solid`. Our implementation can be used directly within any TypeScript or JavaScript application. Furthermore, we provide a script using which queries can be executed from the command line, as shown in Fig. 2.

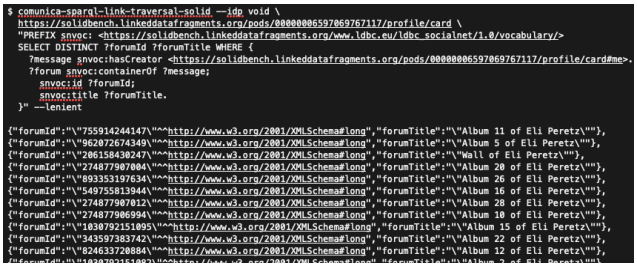

Fig. 2: Executing a SPARQL query from the command line.

4 DEMONSTRATION

In this section, we introduce the environment of our demonstration through a Web-based user interface, after which we discuss our main demonstration scenario.

4.1 User Interface

We demonstrate our traversal-based SPARQL query engine for Solid pods through a Web-based user interface. Since our engine was implemented in TypeScript, it can be transpiled down to JavaScript, and executed client-side within any Web browser. Concretely, we make use of the Comunica jQuery widget *(https://github.com/comunica/jQuery-Widget.js)* to generated a static HTML page that contains our traversal-based query engine, as shown in the figure below. This page is hosted on https://comunica.github.io/comunica-feature-link-traversal-web-clients/builds/solid-default/, and will remain available permanently after the conference.

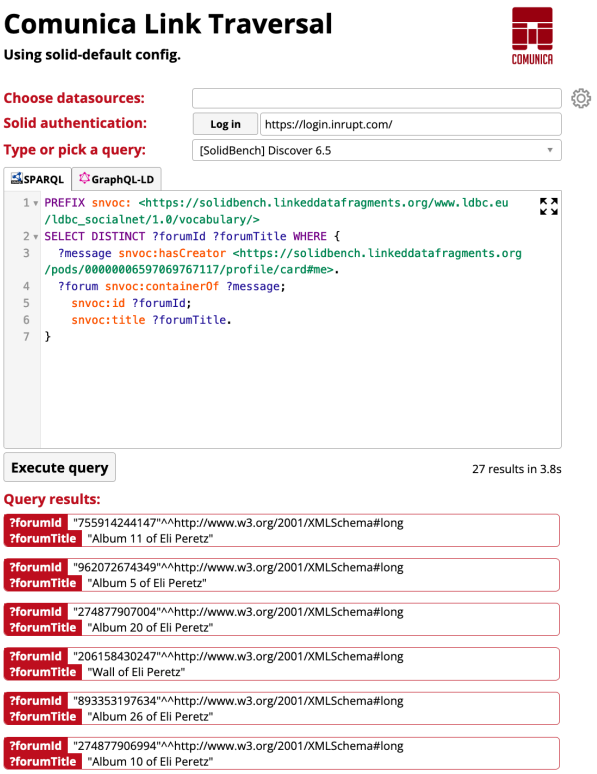

Fig. 3: Web-based user interface of our traversal-based SPARQL query engine.

Our Web-based user interface allows users to write custom SPARQL queries and select seed URLs by selecting datasources from the dropdown-list or typing in custom URLs. If no datasources are selected, the engine will fallback to a query-based seed URL selection approach, where URLs mentioned in the SPARQL query will be considered as seed URLs. Next to writing custom SPARQL queries, the user may also select one of the predefined queries from the dropdown-list. Furthermore, the user can also authenticate with a Solid account by using the "Log in" button.
After inserting or selecting a SPARQL query, the user can click on the "Execute query" button, after which the query engine will execute the query. This execution will take place in a separate Web worker process, as not to halt interactions with the user interface. Each result that is iteratively produced by the query engine will be shown immediately in the scrollable list of query results at the bottom of the page.

4.2 Scenario

Our publicly available Web-based user interface allows users to execute SPARQL queries across any existing Solid pods. Since not everyone may own a Solid pod, our primary demonstration scenario involves an environment in which we have setup a large number of simulated Solid pods.
Concretely, we host 1.531 Solid pods that were generated using the default settings of the SolidBench [14] dataset generator, which consists of 3.556.159 triples spread over 158.233 RDF files across these pods. The SolidBench dataset is derived from the LDBC Social Network Benchmark (SNB) [27], which provides a social network use case, in which people can be friends of each other, create posts, and like and comment on each others posts. Furthermore, we have pre-configured some of the SPARQL queries that SolidBench provides, which can then be executed over this simulated environment.
Our demonstration scenario will start by showing the SPARQL query *"Discover 1.5"* from SolidBench, which will produce all posts that are created by a specific person. For this, we will enable the Network inspection tool tab within the Chrome browser, which allows the audience to see the Resource Waterfall of all HTTP requests that were required to execute the query. This shows how some HTTP requests depend on other requests due to links between them, while other requests may be done in parallel. Fig. 4 shows a screenshot of this scenario.

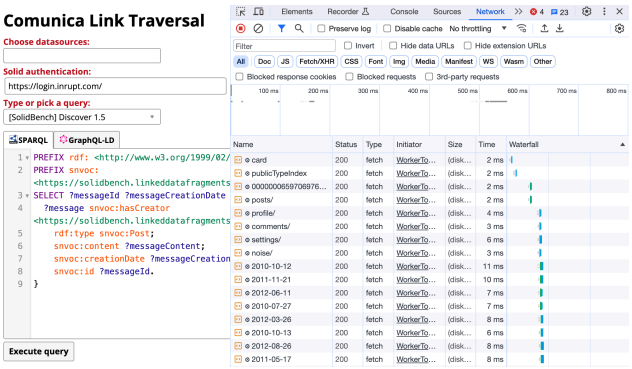

Fig. 4: The Resource Waterfall logs when executing Discover 1.5 from SolidBench.

Since *"Discover 1.5"* will primarily target a single Solid pod, we will also show *"Discover 8.5"*, which targets multiple Solid pods and will return all posts by authors of posts that a given person likes. In contrast to the previous query, *"Discover 8.5"* will traverse across multiple Solid pods, as shown in Fig. 5 Since this happens in a traversal-based manner, all of this happens automatically in the background without requiring any user interaction.

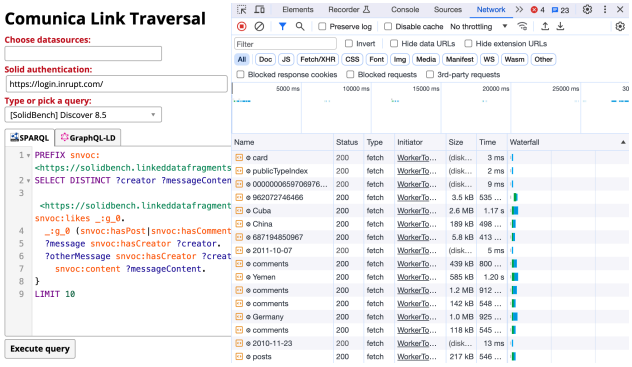

Fig. 5: The Resource Waterfall logs when executing Discover 8.5 from SolidBench.

Besides these 2 queries in the main demonstration scenario, we provide a total of 37 default queries that can be selected in the dropdown-list of queries.

5 CONCLUSIONS

In this article, we discussed the implementation of an open-source SPARQL query engine that is able to query over DKGs within Solid, a permissioned decentralization environment. Our system is based on LTQP techniques that were specifically designed for exploiting the structural properties of the Solid environment.
To demonstrate this system, we provide a Web-based user interface that can be accessed and used by anyone with a modern Web browser. We provide several default SPARQL queries that users can execute across simulated Solid pods, or users may write their own SPARQL queries for execution against real-world Solid pods. Besides this Web-based user interface, our system can be used within any TypeScript or JavaScript application, or via the command-line.
Through this demonstration, we show the effectiveness of traversal-based SPARQL query execution across DKGs. Many queries start producing results in less than a second, which is below the threshold for obstructive delay in human perception [28] in inter-

active applications. Our system and its demonstration provide groundwork for future research in traversal-based query processing over DKGs. In future work, we will investigate further optimizations, which may involve adaptive query planning techniques [29] –which have only seen limited adoption within LTQP [30] and SPARQL query processing [31,32,33]– or enhancements to the link queue [34].

ACKNOWLEDGEMENTS

This research was supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral fellow of the Research Foundation – Flanders (FWO) (1202124N).

REFERENCES

[1] Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).
[2] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G.de, Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., others: Knowledge graphs. Synthesis Lectures on Data, Semantics, and Knowledge. 12, 1–257 (2021).
[3] Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: Proceedings of the 13th International Conference on Database Theory. pp. 4–33. ACM (2010).
[4] Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL Basic Graph Pattern Optimization using Selectivity Estimation. In: Proceedings of the 17th international conference on World Wide Web. pp. 595–604. ACM (2008).
[5] Verborgh, R.: Re-decentralizing the Web, for good this time. In: Seneviratne, O. and Hendler, J. (eds.) Linking the World's Information: A Collection of Essays on the Work of Sir Tim Berners-Lee. ACM (2022).
[6] Bluesky: Bluesky. https://blueskyweb.xyz/ (2023).
[7] Zignani, M., Gaito, S., Rossi, G.P.: Follow the Mastodon: Structure and Evolution of a Decentralized Online Social Network. In: Twelfth International AAAI Conference on Web and Social Media (2018).
[8] Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization Techniques for Federated Query Processing on Linked Data. In: International semantic web conference. pp. 601–616. Springer (2011).
[9] Saleem, M., Ngomo, A.-C.N.: Hibiscus: Hypergraph-based Source Selection for SPARQL Endpoint Federation. In: European semantic web conference. pp. 176–191. Springer (2014).
[10] Görlitz, O., Staab, S.: Splendid: SPARQL Endpoint Federation Exploiting Void Descriptions. In: Proceedings of the Second International Conference on Consuming Linked Data-Volume 782. pp. 13–24. CEUR-WS. org (2011).
[11] Dang, M.-H., Aimonier-Davat, J., Molli, P., Hartig, O., Skaf-Molli, H., Le Crom, Y.: FedShop: A Benchmark for Testing the Scalability of SPARQL Federation Engines. In: International Semantic Web Conference. pp. 285–301. Springer (2023).
[12] Ragab, M., Savateev, Y., Moosaei, R., Tiropanis, T., Poulovassilis, A., Chapman, A., Roussos, G.: ESPRESSO: A Framework for Empowering Search on Decentralized Web. In: International Conference on Web Information Systems Engineering. pp. 360–375. Springer (2023).
[13] Vandenbrande, M., Jakubowski, M., Bonte, P., Buelens, B., Ongenae, F., Van den Bussche, J.: POD-QUERY: Schema Mapping and Query Rewriting for Solid Pods. In: ISWC2023, the International Semantic Web Conference (2023).
[14] Taelman, R., Verborgh, R.: Link Traversal Query Processing over Decentralized Environments with Structural Assumptions. In: Proceedings of the 22nd International Semantic Web Conference (2023).
[15] Mendelzon, A.O., Mihaila, G.A., Milo, T.: Querying the world wide web. In: Fourth International Conference on Parallel and Distributed Information Systems. pp. 80–91. IEEE (1996).
[16] Konopnicki, D., Shmueli, O.: Information gathering in the World-Wide Web: the W3QL query language and the W3QS system. ACM Transactions on Database Systems (TODS). 23, 369–410 (1998).
[17] Chakrabarti, S., Van den Berg, M., Dom, B.: Focused crawling: a new approach to topic-specific Web resource discovery. Computer networks. 31, 1623–1640 (1999).
[18] Batsakis, S., Petrakis, E.G.M., Milios, E.: Improving the performance of focused web crawlers. Data & Knowledge Engineering. 68, 1001–1013 (2009).
[19] Hartig, O., Freytag, J.-C.: Foundations of Traversal based Query Execution over Linked Data. In: Proceedings of the 23rd ACM conference on Hypertext and social media. pp. 43–52. ACM (2012).
[20] Hartig, O.: Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: Extended Semantic Web Conference. pp. 154–169. Springer (2011).
[21] Hartig, O.: SPARQL for a Web of Linked Data: Semantics and computability. In: Extended Semantic Web Conference. pp. 8–23. Springer (2012).
[22] Wilschut, A.N., Apers, P.M.G.: Pipelining in query execution. In: Proceedings. PARBASE-90: International Conference on Databases, Parallel Architectures, and Their Applications. p. 562. IEEE (1990).
[23] Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0. W3C, https://www.w3.org/TR/ldp/ (2015).
[24] Capadisli, S., Berners-Lee, T.: Solid WebID Profile. Solid, https://solid.github.io/webid-profile/ (2022).
[25] Turdean, T.: Type Indexes. Solid, https://solid.github.io/type-indexes/ (2022).
[26] Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a Modular SPARQL Query Engine for the Web. In: Proceedings of the 17th International Semantic Web Conference (2018).
[27] Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D., Boncz, P.: The LDBC social network benchmark: Interactive workload. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 619–630 (2015).
[28] Nielsen, J.: Response times: the three important limits. Usability Engineering. (1993).
[29] Deshpande, A., Ives, Z., Raman, V.: Adaptive query processing. Foundations and Trends\textregistered in Databases. 1, 1–140 (2007).
[30] Hartig, O., Özsu, M.T.: Walking without a Map: Optimizing Response Times of Traversal-based Linked Data Queries (extended version). arXiv preprint arXiv:1607.01046. (2016).
[31] Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: an adaptive query processing engine for SPARQL endpoints. In: International Semantic Web Conference. pp. 18–34. Springer (2011).
[32] Acosta, M., Vidal, M.-E.: Networks of linked data eddies: An adaptive web query processing engine for RDF data. In: International Semantic Web Conference. pp. 111–127. Springer (2015).
[33] Heling, L., Acosta, M.: Robust query processing for linked data fragments. Semantic Web. 1–35 (2022).
[34] Eschauzier, R., Taelman, R., Verborgh, R.: How Does the Link Queue Evolve during Traversal-Based Query Processing? In: Proceedings of the 7th International Workshop on Storing, Querying and Benchmarking Knowledge Graphs (2023).