Revisiting Link Prioritization for Efficient Traversal in Structured Decentralized Environments

Ruben Eschauzier$^{1[0000-0002-6475-806X]}$, Ruben Taelman$^{1[0000-0001-5118-256X]}$, and Ruben Verborgh$^{1[0000-0002-8596-222X]}$

Department of Electronics and Information Systems, Ghent University – imec

Abstract. Decentralized environments distribute personal data across numerous small, independent data sources; a necessity driven by legal and socio-economic constraints that prevent the technologically more convenient central aggregation. Link Traversal-based Query Processing (LTQP) is a query technique that respects these constraints by iteratively discovering and accessing data sources while enabling fine-grained access control. Unfortunately, current LTQP implementations are slow due to limited prior knowledge of queried data and the high volume of HTTP requests required. Prioritizing data sources likely to lead to query-relevant data can improve query result arrival times. However, while link prioritization algorithms have been studied for Linked Open Data (LOD), their performance in structured decentralized environments remains untested. Evaluating this performance is essential to establish a baseline as a reference point for improving future implementations. We formally define the R^3 metric to measure prioritization performance, extend it to continuous efficiency, and account for real-world scenarios. Furthermore, we provide modular and open-source implementations of the prioritization algorithms from the literature. Finally, using the R^3 metric with existing metrics from the literature, we benchmark these link prioritization algorithms in a simulated Solid environment. In this paper, we report the benchmark results, provide a thorough analysis, and lessons learned for future work. We find that existing prioritization algorithms fail to improve performance in structured decentralized environments, with no non-Oracle method outperforming the look-up order produced by a FIFO queue. We conclude that prioritization algorithms have little benefit in a structured decentralized environment, and recommend that research shift to pruning irrelevant links or improving the query plan.

Keywords: Link Traversal-based Query Processing · Benchmarking · Link Prioritization

1 Introduction

While centralizing data can benefit query engine performance, it is often infeasible due to privacy and access control constraints. Decentralized platforms like

Solid [30] and Mastodon [28] promote storing data across many small sources with agreed-upon structures, emphasizing user control over personal data. This shift introduces technical challenges for query engines, which must discover and query numerous small data stores, each with its own access policies.

Centralized querying requires collecting large amounts of personal data, which is restricted by fine-grained access controls. Traditional *federated* approaches [33,31,18] support decentralization but assume a small number (10–100) of known, uniformly accessible sources [7]. Enforcing fine-grained access control policies [10] on such large sources with expressive interfaces significantly impacts performance [25].

In contrast, other decentralized environments involve a large number (thousands) of pre-partitioned permissioned sources that use a wider range of data interfaces consisting of generic HTTP resources [39]. Due to the scale of decentralization, these resources may not all be known at query time. While this does not offer clients the increased flexibility of an expressive query language such as SPARQL, the lower per-resource generation cost makes granular access control less costly. Such interfaces require a federation approach that can query many data sources and interfaces, possibly discovered at runtime, without prior centralization.

Link Traversal-based Query Processing (LTQP) meets these needs by discovering new documents during execution via URIs found in earlier results. Starting from seed documents (user-provided or query-derived), it follows links asynchronously using a FIFO queue. LTQP enables federation at the granularity of the environment, naturally supporting fine-grained access control.

However, the iterative discovery of documents during LTQP makes efficient query processing difficult. Discovering sources on the fly challenges the *optimize-then-execute* query paradigm, as it relies on prior knowledge of the queried data, which is absent in this context. Improving LTQP query optimization and thus performance is critical for enabling more interactive decentralized applications.

Link prioritization algorithms aim to reorder link dereferencing using a priority queue instead of FIFO. By dereferencing links likely to yield query-relevant documents earlier, these algorithms can significantly reduce the time to first result [17] on Linked Open Data, improving perceived performance. This enables interactive applications to display partial results quickly and improve user experience through incremental loading [3,37].

Prioritization algorithms can improve responsiveness in decentralized applications. However, their behavior in structured decentralized environments remains unstudied. Prior work focused on the Linked Open Data (LOD) Cloud, a large, unstructured, and open environment lacking consistent data organization. In contrast, structured decentralized environments enforce predefined schemas, changing the topology of the data web and influencing the performance of the algorithms [17].

In this work, we define structured decentralized environments as decentralized systems where data is organized according to predefined schemas, protocols, or container models, enabling consistent access and reasoning across nodes. For instance, the Solid ecosystem provides structure through the use of Linked Data Platform containers and Type Indexes, allowing clients to make assumptions about data completeness and discoverability. These structural properties are important for effective source selection and query optimization [35]. While our experiments focus on Solid, similar structural properties may arise in other decentralized initiatives that enforce standardized data organization (e.g., ActivityPub [40]).

Evaluating existing prioritization algorithms in structured decentralized environments is essential for identifying their strengths and limitations, providing a baseline for comparison, and ensuring that new methods are built on a clear understanding of what works and what doesn't in these settings. An effective baseline provides a reliable and meaningful standard for comparison. To support robust evaluation, it should also be easily reproducible and built with a modular implementation, enabling ablation studies and facilitating comparisons across different engines and environments. Furthermore, future research on prioritization benefits from modular baseline implementations, facilitating their reuse, extension, and improvement.

Currently, prioritization algorithm implementations in the literature are tied to the SQUIN engine [14], which is no longer maintained and therefore unreliable as a baseline. Moreover, the reported results for these algorithms [17] are influenced by the specific query engine used, its implementation details [11], and other unrelated optimization strategies [13]. As a result, comparing these results to those of new prioritization algorithms implemented in a different engine introduces confounding factors, undermining the validity of the comparison.

To address this, we re-implement existing algorithms as modular packages in Comunica [34], including necessary system changes for (adaptive) prioritization in its iterator-based architecture. These implementations enable fair benchmarking in structured decentralized environments using established metrics and a standard benchmark [35].

To ensure this baseline facilitates performance comparisons across engines and provides deeper insights into the marginal performance of prioritization algorithms, we extend the R^3 metric [9]. The R^3 metric evaluates the marginal performance of prioritization algorithms in an implementation-independent manner, enabling cross-engine comparisons and offering a clearer understanding of prioritization performance without confounding noise.

We build on the previously introduced R^3 metric [9] by formally defining and extending the metric, and then using it in our experiment to measure the marginal performance of prioritization algorithms from the literature. We thereby aim to evolve the metric from a proof-of-concept to a rigorously validated tool for prioritization researchers.

In summary, this paper makes the following contributions:

- **Formal Definition of Link Prioritization Metrics** We mathematically define the R^3 metric introduced in prior work and extend it with alternative versions to account for real-world factors, including weighted dereference events and continuous efficiency.

- **Modular Design and Implementation of Prioritization Algorithms**: We reproduce existing link prioritization algorithms using a modular, iterator-based architecture in a state-of-the-art LTQP engine designed for structured decentralized environments. In addition, we include a new environment-specific prioritization algorithm.
- **Benchmarking and Evaluation of Link Prioritization in Structured Decentralized Environments**: We evaluate the implementation of prioritization algorithms using the decentralized benchmark SolidBench [35], analyzing their performance with existing metrics and newly defined variations of the R^3 metric.

2 Related Work

2.1 Link Traversal-based Query Processing

The seminal work on Link Traversal-based Query Processing (LTQP) [15,14] introduces a method for dynamically expanding and querying a local RDF dataset by recursively dereferencing URIs found in previously dereferenced data. The approach employs an iterator-based pipelining architecture to produce query results continuously. In this framework, each physical operator is implemented as an iterator that consumes input from its predecessor iterators. Before an iterator can process an intermediate result, all URIs contained in that result must be dereferenced and their corresponding RDF data integrated into the local dataset. To avoid blocking query execution during URI lookups, the iterators are designed to defer the processing of intermediate results for which the required URIs have not been dereferenced and instead continue with the next input. Finally, to prevent the dereferencing of an intractable number of URIs, reachability criteria [16] are used to filter out likely irrelevant URIs. A widely adopted criterion, cMatch, restricts traversal to URIs that match a triple pattern in the query, reducing the number of links followed.

The next iteration of LTQP [17] introduces three operator types: a *data retrieval operator (DR-operator)*, a *dispatcher*, and *triple pattern operators (TP-operators)*, one for each query pattern. The DR-operator dereferences URIs, forwarding matching triples to their corresponding TP-operators. TP-operators, inspired by Eddies [1], maintain an internal index of matching triples and probe it for intermediate join results. Once a result has passed through all TP-operators, it is output as a final result. This design enables flexible, adaptive query processing by dynamically adjusting the execution order of TP-operators.

Recent advancements in LTQP for structured decentralized environments [35] combine the iterator-based pipelining architecture with the DR-operator while using structural cues from the target environment. This approach extracts links from structure-indicating predicates, regardless of standard reachability criteria, and employs TypeIndexes [38] to identify URIs pointing to specific resources, such as comments or posts. This effectively implements structurally informed link prioritization. In the context of Solid, these predicates indicate the Linked Data Platform (LDP) [22] structure used in Solid Pods.

2.2 Metrics for Link Traversal-based Query Processing

Database benchmarks have traditionally employed a variety of metrics to evaluate query engine performance, measuring aspects such as query execution efficiency and result correctness. This section examines metrics relevant to query execution performance, specifically metrics usable for measuring link prioritization effectiveness. Table 1 summarizes commonly used benchmark metrics across three key attributes:

Continuous indicates whether the metric evaluates the continuous result production performance of an engine, as opposed to the performance of an engine at a single point.

Algorithm-bound indicates whether the metric isolates the performance of a specific algorithm within a system or evaluates the combination of all algorithms in a system.

Implementation-independent indicates whether the metric reflects performance independent of implementation-specific details and optimizations, rather than being influenced by how an algorithm is implemented.

The Query Response Time metric, commonly used in benchmarks like LUBM [12], measures the total time to complete a query from the moment it is issued. The Berlin SPARQL Benchmark (BSBM) [4] extends this by assessing metrics such as the number of query mixes or specific query types executed per second, along with minimum and maximum query execution times [24]. Diefficiency metrics [2], on the other hand, evaluate continuous query engine performance by calculating the area under the result arrival distribution curve within a specified interval, effectively capturing the result arrival rate, where a higher rate indicates faster result production.

For link traversal engines, metrics such as the time to retrieve the *first*, *last*, and *half of all results* have been widely used [35,39]. Since LTQP delivers results incrementally through its iterator-based approach, optimizing result arrival times is crucial for improving usability. To assess the impact of prioritization algorithms in LTQP, researchers compare the *relative result arrival time* of these algorithms against the *total query execution* time [17], as prioritization primarily affects result arrival rates rather than overall execution time. From Table 1, we observe that these metrics capture both point-in-time and continuous system performance, as well as the behavior of individual algorithms. However, they lack implementation-independence, making direct comparisons across different systems challenging.

3 Method

This section provides a formal mathematical definition of the R^3 metric [9], previously only introduced informally. It also extends the metric to a novel weighted version and introduces a new metric, R^3Cont, to measure continuous prioritization efficiency. Lastly, we outline the prioritization algorithms used for benchmarking.

Cont.	**Alg.**	**Indep.**	**Metric**
			Query Response Time [12]
			Queries per Second [4]
			Query Mixes per Hour [4]
			Min/Max Query Execution time [4,24]
			Overall Runtime [4]
			Time until First/Last Result [17,35]
	✓		Relative Time Until First/Last Result [17]
✓			Diefficiency [2]
	✓	✓	R^3 [9]
✓	✓	✓	$R^3 Cont$ [proposed]

Table 1: Existing metrics tend to assess point-in-time rather than *continuous performance* (**Cont.**) of an entire engine rather than *isolated algorithms* (**Alg.**) of specific implementations rather than being *implementation-independent* (**Indep.**). None capture the desired combination of all three, which our proposal addresses.

3.1 Preliminaries

SPARQL is a pattern-matching query language designed to query the RDF data model. The RDF data model defines triples that describe statements. These triples are formed from three disjoint sets B (blank nodes), L (Literals), and U (URIs), and are of the form $(s, p, o) \in (B \cup U) \times U \times (B \cup L \cup U)$. A triple uses an object o to make a statement about subject s through predicate p. A finite collection of triples forms a labeled directed graph called an RDF graph [32,27]. The simplest form of SPARQL query is a Basic Graph Pattern (BGP) which is defined as a finite set $P = \{tp_1, \dots tp_n\}$ of triple patterns $tp_i \in (V \cup L \cup U) \times (U \cup V) \times (V \cup L \cup U)$, where V is a set of variables. To evaluate SPARQL queries over RDF data, the concept of solution mappings plays a central role. A solution mapping is a partial function $\mu : v \rightarrow (U \cup L)$, where v is the set of variables appearing in the BGP of query Q. Solution mappings associate variables with RDF terms, enabling the identification of matches for the query's graph patterns in the RDF graph.

In the context of Link Traversal, query execution involves dereferencing URIs to retrieve the RDF graphs D embedded in these URIs. From the triples in these graphs, new URIs are extracted based on a reachability criterion c. A reachability criterion is a function $c : T \times U \times \mathcal{B} \rightarrow \{true, false\}$, where T denotes the infinite set of all possible data triples and $\mathcal{B}$ the infinite set of possible BGPs and U as defined above. Reachability criteria determine whether a URI should be considered reachable for a given BGP. An example of a reachability criterion is $cAll$, which is true for all triples. [16]

3.2 Relevant Traversal Path

During LTQP, the engine must dereference all documents contributing to the query result set to complete the query. However, during the traversal, the engine could dereference query-irrelevant documents that contain no triples required to answer the query. By defining the traversal path taken by the engine to dereference all query-relevant documents, we can formally define a metric that measures the efficiency of the path taken by the engine.

First, we must compute the where-provenance [5], the set of URIs to dereference to obtain all data required to answer the query, of each solution mapping μ_n produced by conjunctive query $\mathcal{Q}_c^{P,S}$, with P a BGP, and S a finite set of seed URIs. The evaluation of query $\mathcal{Q}_c^{P,S}$ depends on the subset of *reachable* data during traversal. This *reachable sub-web* $W_c^{S,P}$ [16] denotes the set of discoverable URIs given a reachability criterion, seed URI, and BGP. The evaluation of $\mathcal{Q}_c^{P,S}$ on $W_c^{S,P}$ produces the solution mapping $\mathcal{Q}_c^{P,S}(W)$.

Given where-provenance annotations D_{μ_n} with $\mu_n \in \mathcal{Q}_c^{P,S}(W)$, we can define the set of URIs required to be dereferenced to produce the result as

$$D_{\mathcal{T}} = \bigcup_{\mu_n \in \mathcal{Q}_c^{P,S}(W)} D_{\mu_n}. \tag{1}$$

During query execution, URIs are sequentially dereferenced and added to the queried data. This produces a traversal order of URIs $(W_c^{S,P}, \preceq)$, a totally ordered set with order

$$\forall d_a, d_b \in W_c^{S,P} : d_a \preceq d_b \Leftrightarrow t(d_a) < t(d_b). \tag{2}$$

With $t(d_a)$ the timestamp when the engine finished dereferencing URI d_a. Assuming deterministic HTTP request times, this ordering is produced by applying a traversal order algorithm during query execution. Using this ordering, we define the totally ordered set

$$R_{\mathcal{T}} = (D_{\mathcal{T}}, \preceq) \tag{3}$$

as the query-relevant documents ordered by their dereference time. We use this to define the minimal set of dereferenced documents required to dereference all query-relevant documents:

$$O_{\mathcal{T}} = \{d_i \in (W_c^{S,P}, \preceq) \mid t(d_i) < max\{t(d) \mid d \in D_{\mathcal{T}}\}\}. \tag{4}$$

This set includes all documents with a dereference timestamp smaller than the maximal timestamp among the query-relevant documents. In simpler terms, it represents the traversal path followed by the engine to retrieve all documents necessary for answering the query.

3.3 Optimal Traversal Path for Induced Sub-Web

To compare the performance of link traversal algorithms with an optimal *oracle algorithm* we will first introduce the notion of a traversed topology during

LTQP. During query execution, the engine traverses the sub-web $W_c^{(S,P)}$ of documents belonging to URIs. While the engine will only dereference a URI once, multiple documents can contain references to the same URI. Thus, the engine's traversal path produces a directed (possibly cyclic) unweighted graph $T_{W_c^{(S,P)}}$ with sources S.

In this formulation, we can consider the set $D_{\mathcal{T}}$ to be a set of terminals in $T_{W_c^{(S,P)}}$, then finding the optimal traversal path for the engine is similar to finding the minimum directed Steiner tree.

The Steiner tree problem in directed graphs is defined as follows [6,19]. For a directed graph $G = (V, A)$, with $c(a)$ the cost of arc $a \in A$, root node $r \in V$, and terminals $T \subseteq V$, the objective is to find the minimum cost sub-graph starting at r and spanning all vertexes T.
The cost for such a sub-graph $X = (V', A')$ in a directed graph is given as

$$C(X) = \sum\nolimits_{a \in A'} c(a), \tag{5}$$

where $c(a) = 1$ for all $a \in A$. The topology needs a singular root node to formulate traversed topologies as a Steiner tree for directed graph problem. However, topologies can have multiple root nodes corresponding to the seed documents supplied to the query. So, we can add a new root node $\hat{r}$ to our traversed topology and create arcs $\{(\hat{r}, s) \mid s \in S\}$ with costs 0. So, for our problem, we have to find a directed Steiner tree for directed graph $T_{W_c^{(S,P)}}$, root $\hat{r}$, and terminals $D_{\mathcal{T}}$.

3.4 Relevant Retrieval Ratio R^3

Building on the optimal traversal path X and the realized minimal engine traversal path $O_{\mathcal{T}}$ introduced earlier, we now provide the *first formal definition* of the R^3 metric. Unlike the previous definition [9] that relies on informal intuition, our definition mathematically defines the relative performance of the engine compared to the optimal approach. R^3 is formally defined as:

$$R^3 = \frac{C(X)}{\mid O_{\mathcal{T}} \mid} \tag{6}$$

For this metric, higher is better, as this signifies that the traversal path generated by the engine is closer to optimal. Figure 1 illustrates an example topology, the optimal traversal path, and the selected traversal path. For these paths, the R^3 metric is calculated as $R^3 = \frac{3}{6} = 0.5$, indicating that the cost of the selected path is twice that of the optimal path.

The R^3 metric captures the traversal path followed during query execution over the Linked Data Web. This path is largely determined by the engine's prioritization and traversal strategy, rather than by engine-specific optimizations or query planning algorithms. As a result, even if execution times vary across engines, R^3 still reveals the relative performance of traversal strategies, enabling fair comparison across different engine implementations.

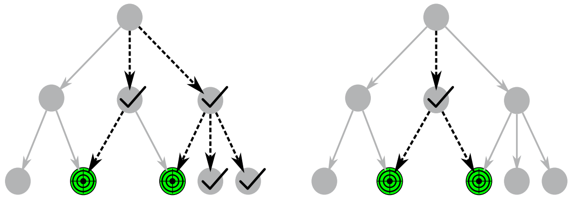

Fig. 1: The traversal on the left follows an inefficient dereference path (checkmarks) to the target documents (green), compared to the optimal path on the right. The R^3 metric captures this inefficiency.

3.5 Weighted Relevant Retrieval Ratio R^3

The previous formulation of the R^3 metric assumes equal costs for each traversal step. However, this approach overlooks real-world factors that impact query execution, such as HTTP request time, which can be influenced by server congestion or significant physical distance between the client and server.

To account for these differences in the traversed topology graph we propose to weigh edges by factors influencing the real-world notion of optimal traversal, like HTTP request time. The optimal sub-graph found by Steiner tree solvers then reflects the real-world costs of traversal. We define the ***weighted*** R^3 **metric** as follows:

$$\hat{R}^3 = \frac{C(X)}{C(O_{\mathcal{T}})} \tag{7}$$

Figure 2 shows how this weighted $\hat{R}^3$ metric can alter the optimal path. In this example, one node experiences a significant delay in responding to an HTTP request. Consequently, the optimal path changes to a route with more hops but a shorter overall HTTP request time. Due to the change in weights, the previously optimal path, which had $R^3 = 1$, now has $\hat{R}^3 = \frac{1+1+3+3}{5+3+3} = 0.73$.

As is often the case, there is no universally optimal method for weighting; the choice depends on the specific optimization target. For instance, minimizing the size of HTTP request responses becomes crucial in environments with limited bandwidth.

3.6 Continuous Efficiency of Link Prioritization Algorithms

In earlier sections, we examined metrics evaluating the overall performance of prioritization algorithms against the optimal approach. However, since LTQP produces results incrementally as they become available, it is important to assess the algorithm's continuous performance during query execution.

To achieve this, we propose adapting the diefficiency metrics [2], which measure the area under the answer distribution function to assess the continuous efficiency

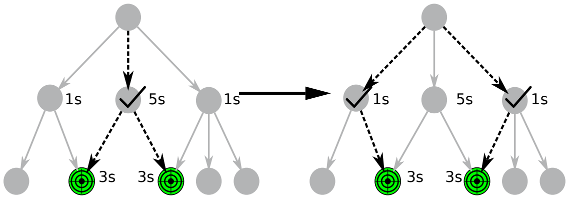

Fig. 2: By considering HTTP request latency, the weighted R^3 metric ensures that the computed traversal path corresponds to the real-world optimum.

of query engines. This function is constructed using an answer trace, defined as a sequence of pairs $A_r = (t_1, \mu_1), \ldots, (t_n, \mu_n)$, where each pair records the timestamp t_i when the i-th answer is produced, for $1 \leq i \leq n$. For non-blocking approaches like LTQP, the answer distribution function represents the number of results produced over time, with linear interpolation applied between arrivals.

We modify the answer trace to evaluate the continuous efficiency of link prioritization algorithms in retrieving documents required for query answers. The new trace, $A_d = (t_1, d_i), \ldots, (t_n, d_n)$, instead records the timestamps when the required documents for a new answer are dereferenced. Based on this trace, we calculate the new R^3Cont at k results metric as

$$R^3Cont@k = \int_0^{t_k} A_d(x)dx, \tag{8}$$

where t_k denotes the time when the documents needed for the first k results are retrieved. Note that this function is analogous to the *Dief@k* metric [2]. Thus, $R^3Cont@k$ measures the continuous efficiency of the traversal phase of the LTQP engine, with lower values being better

3.7 Existing Link Prioritization Algorithms

This paper establishes a baseline for link prioritization algorithms in structured decentralized environments. We implement existing algorithms [17] in Comunica [34] and release them as modular, user-friendly packages. Unlike the TP-operator-based model [17], our implementation uses the more widely adopted iterator-based model [20,14,35,23]. To enable adaptive prioritization, we wrap each iterator produced during query execution with an event-driven statistics emitter, which feeds into the prioritization strategy. Adopting this model facilitates comparisons across query engines and promotes broader applicability.

We briefly explain the implemented algorithms, with detailed descriptions available in the original paper [17].

Non-Adaptive Non-adaptive algorithms use a pre-defined prioritization strategy. These algorithms include *depth-first* and *breadth-first* prioritization.

Additionally, *random* prioritization assigns a random priority between 0 and 10 to each new link. Finally, we include an implementation of a theoretical *oracle* algorithm that uses the result contribution counter of each document. This counter is propagated across all nodes along the shortest path from the root to the query-relevant document.

Graph-based For graph-based prioritization, the engine iteratively builds a directed traversed topology graph $G = (V, E)$, representing the queried sub-web. Links are prioritized by applying vertex scoring algorithms to this topology. We examine two such algorithms: the first sets the priority of links equal to their *in-degree*, the second uses the vertex's *PageRank* score [26].

Result-based Result-based prioritization uses the traversed topology of graph-based prioritization methods. The engine tracks the why-provenance of results and increments the result contribution counter (RCC) of the associated documents whenever a new result is produced. Based on these RCC values, [17] introduces vertex scoring functions using the first- and second-degree in-neighborhoods of documents, defined as:

$$\mathrm{in}^1(v) = \{v' \in V \mid \langle v', v \rangle \in E\}, \tag{9}$$

$$\mathrm{in}^2(v) = \mathrm{in}^1(v) \cup (\bigcup\nolimits_{v' \in \mathrm{in}^1(v)} \mathrm{in}^1(v')). \tag{10}$$

For a vertex v with an RCC score of $rcc(v)$, four vertex scoring functions are defined for $k \in 1, 2$:

$$\text{rcc-k}(v) = \sum\nolimits_{v' \in \mathrm{in}^k(v)} \mathrm{rcc}(v'), \tag{11}$$

$$\text{rel-k}(v) =\mid \{v' \in \mathrm{in}^k(v) \mid rcc(v') > 0\} \mid . \tag{12}$$

The prioritization approaches based on these scoring functions are referred to as *rcc1*, *rcc2*, *rel1*, and *rel2*, respectively.

Intermediate result-based Whereas the previous approach relies on full solutions, intermediate solutions can also guide prioritization decisions. The methods *IS* and *ISdcr* first determine the number of triple patterns satisfied by an intermediate result, storing this count as *cnt*. They then iterate over each binding in the intermediate result. If a bound value corresponds to an IRI in the look-up queue with a priority lower than *cnt*, its priority is updated to *cnt*. For *IS*, initial priorities are set to 0, whereas for *ISdcr*, they are initialized to one less than the final priority of the parent document.

Hybrid Approaches Hybrid prioritization approaches combine RCC-based and intermediate solution-based methods. For each vertex in the traversed topology, these methods track both the RCC and the *IS*-score. The priority for a vertex is then calculated by multiplying its *IS*-score with the RCC-based score, depending on the specific RCC scoring method used. The resulting approaches are named *is-rcc1*, *is-rcc2*, *is-rel1*, and *is-rel2*.

TypeIndex-based Finally, we introduce a simple approach to prioritize links in a *TypeIndex*[38], which points to files containing resources of a specific type. Listing 1.1 shows an example of a TypeIndex pointing to the location of Comments. This algorithm is an example of prioritization based on environment-specific attributes of a structured decentralized environment.

Listing 1.1: Example TypeIndex Content

```
<#bq1r5e> a solid:TypeRegistration;
  solid:forClass <http://example.org/Comment>;
  solid:instanceContainer </public/comments/>.
```

4 Evaluation

This section outlines the experiments used to evaluate link prioritization. We measure time to first and last result, excluding total execution time, since prioritization doesn't affect overall workload. Metrics include $R^3Cont@k$ and $Dief@k$ (with k as total results), as well as both unweighted and HTTP time-weighted R^3 scores. In addition, we compare result counts relative to breadth-first prioritization to account for timeouts, which can cause fewer results to be produced. The goals are to assess the effectiveness of link prioritization in structured decentralized settings, provide a baseline of prioritization performance, and isolate algorithmic performance from implementation overhead. Experiments are run on a dual Hexacore Intel E5645 (2.4 GHz) with 8 GB allocated to Node.js.

4.1 Benchmark

We use the SolidBench benchmark [35], the state-of-the-art benchmark for structured decentralized environments, to evaluate the prioritization algorithms. SolidBench simulates a social network application, including users, posts, comments, forum posts, and more. The benchmark uses a single ontology to describe the relations. Furthermore, data is fragmented to represent Solid pods, where all data related to a specific user is stored in a single pod. Using the default settings, the benchmark generates 158,233 RDF files distributed across 1,531 data vaults, containing a total of 3,556,159 triples.

The benchmark provides three query template types: *discover*, *short*, and *complex.* In this paper, we focus on the *discover* queries, as all complex queries exceed the timeout threshold, and short queries either do not traverse to new pods or time out. Discover queries primarily test various query engine bottlenecks [35], such as n hop link-traversal, irrelevant HTTP request filtering, and heterogeneous within-pod data fragmentation strategies. The timeout is set to 180 seconds, and each query is executed 15 times in our experiment to ensure the reliability and repeatability of the results.

4.2 Results

We report the relative arrival times of the first and last results (*1st* and *Cmpl*) and the R^3 metric. Additionally, we show the number of results produced relative to the breadth-first baseline. Figure 3 presents *1st*, *Cmpl*, and R^3 for 17 algorithms across four representative query templates; the full results are provided in the supplementary material. Overall, no algorithm consistently outperforms the baseline on *1st* or *Cmpl*. Apparent gains for templates interactive-discover 7 and 8 are due to timeouts leading to fewer results, skewing the metrics. The only exception is the oracle, which shows slight improvements in result arrival times.

Table 2 reports the percentage of queries performing 10% better or worse than breadth-first traversal. Overall, *1st* and *Cmpl* are generally worse for the prioritization algorithms compared to the baseline. As seen from the figure, the oracle is the only algorithm to improve on the baseline.

	$1st$		$Cmpl$		R^3		$\hat{R}^3Http$		$Dief$		R^3Cont	
	better	worse	better	worse	better	worse	better	worse	better	worse	better	worse
depth-first	15.7	17.1	14.3	17.1	7.5	12.5	7.5	25.0	25.0	27.5	25.0	30.0
random	4.3	68.6	5.7	74.3	15.0	15.0	25.0	17.5	32.5	35.0	27.5	25.0
in-degree	15.7	27.1	11.4	24.3	5.0	12.5	15.0	20.0	30.0	22.5	27.5	20.0
pagerank	12.9	27.1	11.4	24.3	7.5	7.5	10.0	20.0	32.5	25.0	25.0	17.5
rcc-1	10.0	41.4	10.0	40.0	12.5	10.0	12.5	20.0	25.0	22.5	22.5	35.0
rcc-2	7.1	45.7	7.1	50.0	10.0	15.0	15.0	17.5	30.0	25.0	17.5	42.5
rel-1	10.0	44.3	12.9	44.3	7.5	7.5	12.5	10.0	30.0	27.5	20.0	40.0
rel-2	7.1	45.7	8.6	47.1	10.0	15.0	15.0	22.5	27.5	32.5	25.0	37.5
is	11.4	35.7	12.9	31.4	10.0	**2.5**	20.0	**7.5**	17.5	35.0	32.5	35.0
isdcr	2.9	32.9	5.7	28.6	12.5	15.0	12.5	22.5	25.0	32.5	17.5	62.5
is-rcc-1	4.3	57.1	5.7	60.0	7.5	5.0	10.0	17.5	25.0	27.5	15.0	50.0
is-rcc-2	5.7	62.9	8.6	68.6	12.5	5.0	20.0	17.5	25.0	27.5	12.5	52.5
is-rel-1	7.1	64.3	5.7	61.4	12.5	12.5	17.5	25.0	32.5	**15.0**	20.0	40.0
is-rel-2	5.7	61.4	7.1	64.3	12.5	5.0	17.5	17.5	**37.5**	27.5	17.5	50.0
type-index	15.7	18.6	10.0	15.7	12.5	20.0	15.0	25.0	27.5	30.0	35.0	35.0
oracle	**18.6**	**11.4**	**18.6**	**12.9**	**25.0**	**2.5**	**32.5**	12.5	30.0	30.0	**72.5**	**10.0**

Table 2: Percentage of queries for which a prioritization algorithm performs at least 10% better or worse than the breadth-first baseline. Most algorithms do not consistently improve point-in-time or continuous result arrival times. This limitation is underscored by the R^3 metrics, which show the limited effectiveness of prioritization algorithms even when decision-making overhead is excluded.

We use the R^3 metric to analyze why prioritization algorithms fail to improve performance. As shown in Figure 3 and Table 2, most approaches do not significantly change the engine's traversal efficiency; fewer than 25% of queries show notable differences in unweighted R^3. While the HTTP-weighted $\hat{R}^3$ shows greater variation, this is largely due to noise from our simulated network environment. When traversal paths remain unchanged, prioritization adds overhead without benefit, leading to reduced performance. The oracle improves traversal in 25% of queries and continuous retrieval in 72.5%, but this rarely translates to faster result arrival in practice. This points towards the bottleneck of result production

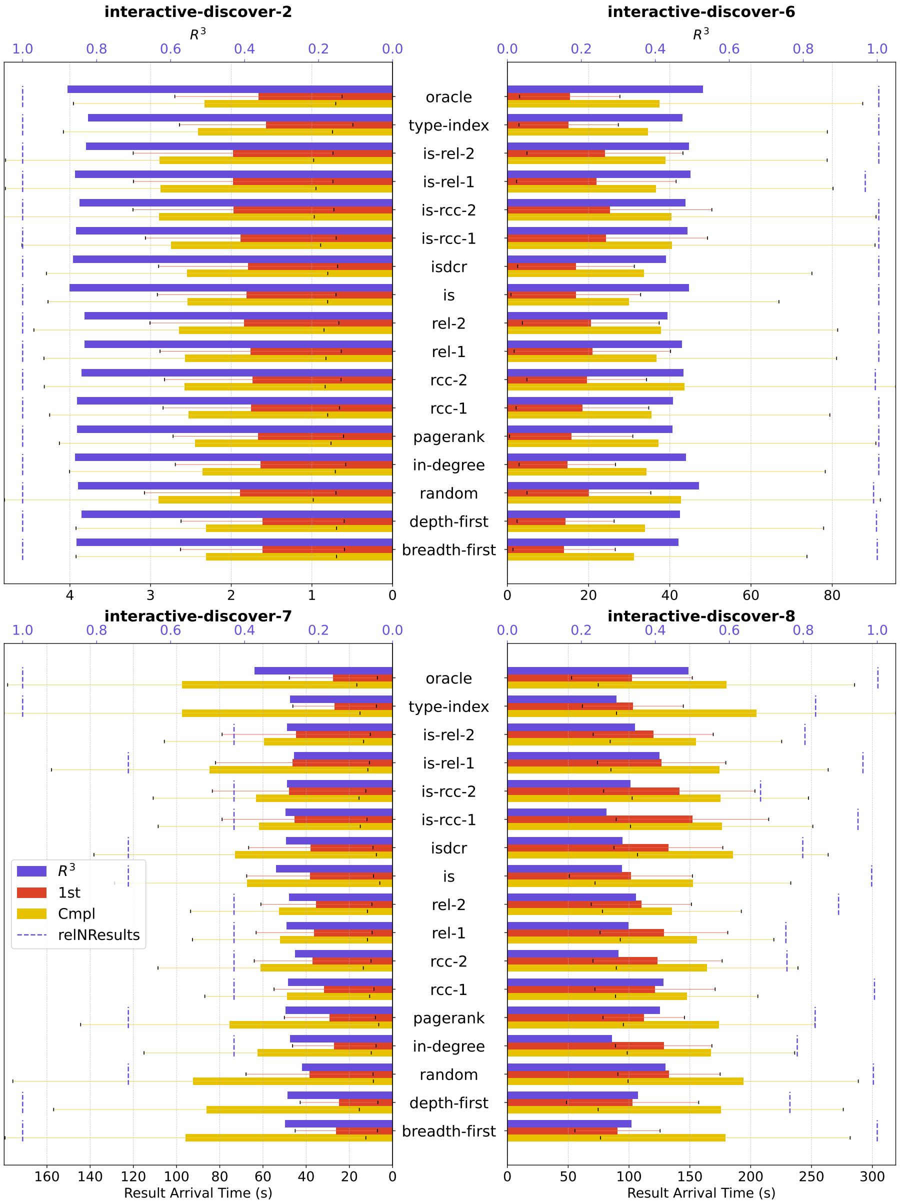

Fig. 3: Result arrival times (first/last), their standard deviations, R^3 scores (higher is better), and result counts relative to breadth-first prioritization. Link prioritization yields no meaningful improvement, often performing 0–20% worse, with similar R^3 values across methods.

lying elsewhere for LTQP. Overall, even a theoretically optimal prioritization has a limited effect in structured decentralized settings like Solid.

Further investigation identifies three key factors influencing these findings:

Query Scope and Structure Many queries, such as discover 1 to 6, target data related to a single individual (and thus a single pod). In such cases, TypeIndexes [35], which act as an index for a pod by directly linking to potentially relevant data, efficiently direct the engine to the correct file containing relevant data, minimizing traversal complexity. These TypeIndexes are found in the seed document in the query, so their links are naturally added early to the queue. As a consequence, TypeIndex prioritization has little effect. We expect similar behavior for any other indexes present in a Solid pod, like a shape index [36]. While queries exist that target multiple pods, these often time out and are thus difficult to analyze.

Solid Environment Design Link prioritization largely focuses on identifying the appropriate LDP container for a query within the Solid environment. Since related data is typically grouped within a single LDP container, the presence of a correctly configured TypeIndex reduces the need for traversing the entire pod to find the relevant data LDP container, simplifying the prioritization problem even when multiple pods are involved in the query.

Link Queue Dynamics During discover queries [8], the link queue is often emptied quickly. Even when an algorithm assigns the correct priority to a link, its impact is negligible if no alternative links are available in the queue for concurrent dereferencing.

Query Plan Bottleneck Previous research [35] shows that link traversal join plans are suboptimal. In cases where the query execution time is dominated by result processing, link prioritization will have a negligible impact on result arrival times.

These findings suggest that link prioritization has a limited impact on link traversal performance in structured, decentralized environments. Even when prioritization is nearly optimal, it yields only modest performance gains. Therefore, efforts to optimize the traversal phase of LTQP should instead focus on strategies such as pruning irrelevant data sources [36]. To rigorously evaluate these strategies, a broader set of feasibly executable queries that traverse multiple pods is required. The short and complex queries of SolidBench do not satisfy these requirements, as their complexity makes them unfeasible to execute. Such benchmarks would help researchers improve their algorithms step by step, without treating performance as simply a matter of timing out or completing the query.

5 Conclusion

In this paper, we provided modular implementations of link traversal prioritization algorithms from the literature, enabling researchers to extend these approaches with minimal effort. Additionally, we benchmarked these algorithms in a previously unexplored context: structured decentralized environments. To further solidify this benchmark as a baseline for future research, we formally introduced, extended, and tested the R^3 metric.

Our findings show that prioritization algorithms do not significantly improve result arrival times compared to the baseline in structured decentralized environments. The only measured improvement came from a theoretical oracle algorithm, but even those gains were small. Therefore, future optimization efforts in LTQP should focus on strategies that either improve the query plan or prune irrelevant data sources. To evaluate these approaches more effectively, a broader set of executable queries spanning multiple pods is needed to better benchmark system performance when traversing many links.

Supplemental Material Statement: The main repository with links to all code needed to replicate the results is available on GitHub: `https://github.com/RubenEschauzier/r3-metric-data-processor`. The raw experimental data and full version of the paper's figure can be found on Zenodo: `https://zenodo.org/records/15372999`. The query templates used to benchmark the algorithms are found on GitHub: `https://github.com/SolidBench/SolidBench.js/tree/master/templates/queries`

Acknowledgements: This research was supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral researcher at the Research Foundation – Flanders (FWO).

References

1. Maribel Acosta and Maria-Esther Vidal. Networks of linked data eddies: An adaptive web query processing engine for rdf data. In *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I 14*, pages 111–127. Springer, 2015.
2. Maribel Acosta, Maria-Esther Vidal, and York Sure-Vetter. Diefficiency metrics: measuring the continuous efficiency of query processing approaches. In *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II 16*, pages 3–19. Springer, 2017.
3. Osmond C Aniebo. *Quantitative Assessment of the Relationship between e/m-Commerce Site Access Speed and Consumer Adoption-Retention Rate*. PhD thesis, Northcentral University, 2020.
4. Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):1–24, 2009.
5. Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*, pages 316–330. Springer, 2001.
6. Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.
7. Minh-Hoang Dang, Julien Aimonier-Davat, Pascal Molli, Olaf Hartig, Hala Skaf-Molli, and Yotlan Le Crom. Fedshop: A benchmark for testing the scalability of sparql federation engines. In *International Semantic Web Conference*, pages 285–301. Springer, 2023.

8. Ruben Eschauzier, Ruben Taelman, and Ruben Verborgh. How does the link queue evolve during traversal-based query processing? In *QuWeDa/MEPDaW@ ISWC*, pages 26–33, 2023.
9. Ruben Eschauzier, Ruben Taelman, and Ruben Verborgh. The r3 metric: Measuring performance of link prioritization during traversal-based query processing. In *Proceedings of the 16th Alberto Mendelzon International Workshop on Foundations of Data Management*, September 2024.
10. Beatriz Esteves, Harshvardhan J Pandit, and Víctor Rodríguez-Doncel. Odrl profile for expressing consent through granular access control policies in solid. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 298–306. IEEE, 2021.
11. Mathieu Fourment and Michael R Gillings. A comparison of common programming languages used in bioinformatics. *BMC bioinformatics*, 9:1–9, 2008.
12. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.
13. Olaf Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *Extended Semantic Web Conference*, pages 154–169. Springer, 2011.
14. Olaf Hartig. Squin: a traversal based query execution system for the web of linked data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1081–1084, 2013.
15. Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing sparql queries over the web of linked data. In *The Semantic Web-ISWC 2009: 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings 8*, pages 293–309. Springer, 2009.
16. Olaf Hartig and Johann-Christoph Freytag. Foundations of traversal based query execution over linked data. In *Proceedings of the 23rd ACM conference on Hypertext and social media*, pages 43–52, 2012.
17. Olaf Hartig and M Tamer Özsu. Walking without a map: Ranking-based traversal for querying linked data. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*, pages 305–324. Springer, 2016.
18. Lars Heling and Maribel Acosta. Federated sparql query processing over heterogeneous linked data fragments. In *Proceedings of the ACM Web Conference 2022*, pages 1047–1057, 2022.
19. Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.
20. Günter Ladwig and Thanh Tran. Sihjoin: Querying remote and local linked data. In *Extended Semantic Web Conference*, pages 139–153. Springer, 2011.
21. Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. Scalable multi-query optimization for sparql. In *2012 IEEE 28th International Conference on Data Engineering*, pages 666–677. IEEE, 2012.
22. Arnaud J Le Hors and Steve Speicher. The linked data platform (ldp). In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1–2, 2013.
23. Daniel P Miranker, Rodolfo K Depena, Hyunjoon Jung, Juan F Sequeda, and Carlos Reyna. Diamond: A sparql query engine, for linked data based on the rete match. *AImWD 2012*, page 7, 2012.
24. Gabriela Montoya, Maria-Esther Vidal, Oscar Corcho, Edna Ruckhaus, and Carlos Buil-Aranda. Benchmarking federated sparql query engines: Are existing testbeds enough? In *The Semantic Web–ISWC 2012: 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II 11*, pages 313–324. Springer, 2012.

25. Ankur Padia, Tim Finin, Anupam Joshi, et al. Attribute-based fine grained access control for triple stores. In *3rd Society, Privacy and the Semantic Web-Policy and Technology workshop, 14th International Semantic Web Conference*, 2015.
26. Lawrence Page. The pagerank citation ranking: Bringing order to the web. Technical report, Technical Report, 1999.
27. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.
28. Aravindh Raman, Sagar Joglekar, Emiliano De Cristofaro, Nishanth Sastry, and Gareth Tyson. Challenges in the decentralised web: The mastodon case. In *Proceedings of the internet measurement conference*, pages 217–229, 2019.
29. Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. A fine-grained evaluation of sparql endpoint federation systems. *Semantic Web*, 7(5):493–518, 2016.
30. Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. Solid: a platform for decentralized social applications based on linked data. *MIT CSAIL & Qatar Computing Research Institute, Tech. Rep.*, 2016, 2016.
31. Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10*, pages 585–600. Springer, 2011.
32. Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th international conference on database theory*, pages 4–33, 2010.
33. Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10*, pages 601–616. Springer, 2011.
34. Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. Comunica: a modular sparql query engine for the web. In *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17*, pages 239–255. Springer, 2018.
35. Ruben Taelman and Ruben Verborgh. Link traversal query processing over decentralized environments with structural assumptions. In *International Semantic Web Conference*, pages 3–22. Springer, 2023.
36. Bryan-Elliott Tam, Ruben Taelman, Pieter Colpaert, and Ruben Verborgh. Opportunities for shape-based optimization of link traversal queries. *arXiv preprint arXiv:2407.00998*, 2024.
37. Monideepa Tarafdar and Jie Zhang. Determinants of reach and loyalty—a study of website performance and implications for website design. *Journal of Computer Information Systems*, 48(2):16–24, 2008.
38. T Turdean. Type indexes. solid. `https://solid.github.io/type-indexes/`, 2022.
39. Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: a low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37:184–206, 2016.
40. W3C. Activitypub: A decentralized social networking protocol. `https://www.w3.org/TR/activitypub/`, January 2018. W3C Recommendation.