Traveling with a Map: Reducing the Search Space of Link Traversal Queries Using RDF Shapes

Journal Title
XX(X):1–16

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Bryan-Elliott Tam[1], Ruben Taelman[1], Joachim Van Herwegen[1] and Pieter Colpaert[1]

Abstract

The centralization of web information raises legal and ethical concerns, particularly in social, healthcare and education applications. Decentralized architectures, offer a promising alternative, but efficient query processing remains a challenge. Link Traversal Query Processing (LTQP) enables querying across decentralized networks but suffers from long execution times and high data transfer due to excessive HTTP requests. We propose a shape-based pruning approach that relies on *shape indexes* and a *query-shape subsumption* algorithm to reduce the search space and consequently, the number of HTTP requests. We formalize this approach as a link pruning mechanism for LTQP and evaluate its effectiveness on social media queries using the SolidBench benchmark across multiple evaluation metrics. Our results show that shape-based pruning improves query execution time and reduces network usage up to 7 times compared to the state of the art, in exchange of a minor increase in the number of triples per shape-index instance. This work demonstrates the potential of shape-based metadata for optimizing LTQP queries in decentralized knowledge graphs, going beyond its traditional use on data validation.

Keywords

Linked Data, Link Traversal Query Processing, RDF data shapes, Decentralization, Data summarization

Introduction

Multiple studies have highlighted problems of ownership, threats to democracy, reinforcement of inequality, and antagonism between users and owners of social web applications, which rely on highly centralized data management systems [64, 15, 53, 40]. Yet, several authors consider decentralizing data over the web an insufficient solution [40, 15]; although it is an integral component of initiatives focused on data sovereignty. Linked Data and knowledge graph (KG) [35] can be considered technical contributions toward the development of a decentralized web of data. However, SPARQL, the standard query language for RDF knowledge graphs, is predominantly performed in centralized environments, partly due to the more mature understanding of query optimization in such settings.

Link Traversal Query Processing (LTQP) [31] is a query paradigm designed for querying unindexed, Decentralized Knowledge Graphs (DKGs) on the web, by leveraging the descriptive power of IRI dereferencing. LTQP involves recursively dereferencing IRIs, dynamically discovering and storing triples from the documents associated with those IRIs in an internal triple store, thereby expanding the engine's underlying knowledge base during query execution. The main difficulty of LTQP is the large domain of exploration, which leads to a high number of HTTP requests as demonstrated by Hartig and Özsu [33]. From another perspective, Taelman and Verborgh [59] showed that in Decentralized Environments with Structural Properties (DESPs), it is possible to attain query completeness for various types of practical queries within acceptable execution times for the context of social media applications [45]. Structural properties ensure data discoverability, which in turn helps guarantee result completeness.

In practice, DESPs emerge in various contexts, such as social networks [59], the publication of sensor data [63], among others. The work of Taelman and Verborgh [59] suggests that various optimizations are feasible for LTQP in decentralized environments with structural properties, in contrast to the more pessimistic conclusion of Hartig and Özsu [33].

In general, the World Wide Web lacks a structure that query engines can exploit for optimization. Any document can be published anywhere, with no standard index or trust mechanism to guide discovery. However, within specific *subwebs*, defined as subsections of the web controlled by particular data providers, implicit or explicit data structures may emerge, which query engines can leverage [9]. In this work, we extend a dataset summarization approach for decentralized environments known as the *shape index* [62]. We apply this approach to enable link pruning within LTQP, removing links that are not relevant to the query, based on an analysis of RDF data shapes and the user's query. The analysis is performed by conducting a query-shape subsumption check to determine whether a resource conforming to a given shape is relevant to a query. Our approach assumes a DKG composed of subwebs, each hosted by data providers and containing shape indexes. A

[1]Universiteit Gent, Ghent, Belgium

Corresponding author:
Bryan-Elliott Tam
Email: {firstname.lastname}@ugent.be

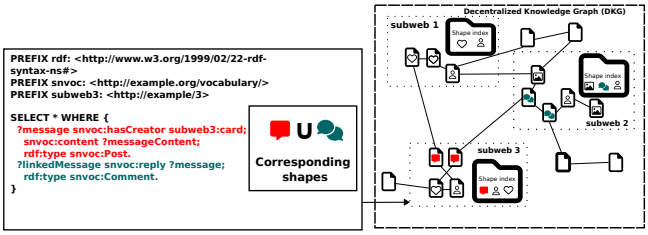

Figure 1. Given that the resources of a DKG are indexed with a shape index, a query engine can dereference a subset of the network. The nodes represent RDF resources, while the edges represent IRIs linking one resource to another. Each subweb has a shape index that maps shapes, represented by icons, to RDF resources by embedding the icon within the node. The query engine starts its query at subweb 3, and the relevant query resources in a subweb are identified with a black node.

subweb that exposes a shape index enables the query engine to narrow its search domain by identifying resources that are certainly non-query-relevant. This is particularly useful when only a subset of a subwebs is relevant to a given query, for example, in social media applications where it is rare to query all information about a user. To illustrate this, we present the example in Figure 1, which depicts a network of three social media user subwebs, each with its own shape index, as well as resources located outside these subwebs. The feature query aims to retrieve posts from subweb 3, along with all associated replies. Our pruning strategy allows the query engine to explore only the relevant parts of the network, guided by the shapes associated with the resources and the structure of the query. The process begins with the engine dereferencing the shape index of subweb 3 and performing a query-shape subsumption check, which determines that only resources containing posts needs to be accessed in this subweb. It then checks the shape index of subweb 1 due to the existing link towards it, where the subsumption check reveals no resources relevant to the query. Next, it examines the shape index of subweb 2 and identifies resources containing comments (i.e., replies) which are relevant. Finally, the engine dereferences all reachable resources outside the subwebs that are linked to these relevant comments.

Our contributions are as follows:

(i) An introduction and formalization of *link pruning* for LTQP

(ii) A shape-based pruning approach for LTQP using RDF data shapes

(iii) A novel network abstractions of *subwebs* and *decentralized knowledge graphs* (DKGs) for LTQP,

(iv) A novel *query-shape subsumption algorithm* for assessing data source relevance

(v) An open-source implementation in the Comunica SPARQL query engine [61]

(vi) An extensive experimental evaluation using the SolidBench benchmark.

Concretely in this work, we ask the following research question: **Can LTQP use shape-based pruning in DKG networks to reduce query execution time while preserving result completeness?** To address this question, we propose the following hypotheses:

H1 Shape indexes reduce the number of non-contributing data sources retrieved and the query execution time.

H2 The execution time of a query-shape subsumption algorithm is negligible in social media applications.

H3 Stricter shape constraints lead to a greater reduction in HTTP requests.

H4 Querying a network with more *complete* shape indexes results in faster query execution.

H5 Performance gain can be acquired in networks with less shape index information.

H6 There is a linear relationship between the reduction in the number of HTTP requests and the decrease in query execution time.

The remainder of this paper is organized as follows: we first review the related work and introduce the preliminaries. Next, we present our approach, followed by a description of the experimental setup and a discussion of the results. Finally, we conclude with the conclusion.

Related Work

In this section, we review related work on LTQP, RDF data shapes, and source selection in decentralized querying.

Link Traversal Query Processing

LTQP is a SPARQL querying paradigm that answers queries by exploring the Web using the *follow-your-nose* principle [33]. It belongs to the family of decentralized SPARQL querying paradigms. LTQP fundamentally differs from federated querying because the "federation" is formed during querying and it expands dynamically as the query is processed. Thus, many optimization techniques used in federated querying either do not work in the context of LTQP or remain unexplored. LTQP also differs from querying Linked Data Fragments (LDF) interfaces [5, 6, 30], as LDF operates over fragments of a known, local, finite KG, whereas LTQP dynamically discovers new, remote KGs during query processing. In summary, LDF methods such as Triple Pattern Fragments (TPF) [65] focus on the efficient partitioning of a known dataset, while LTQP focuses on the efficient discovery of an unknown dataset.

The main challenge of LTQP is the Web's open-ended nature leading to large search spaces. Completeness in LTQP is defined by the traversal of a well-defined set of links [31]. The first method employed to define this set was the reachability criteria [31], which are boolean functions that determine whether a link should be dereferenced. Building on this, the theoretical query language LDQL [34] was introduced, separating the traversal definition from the query definition. Further advancements include the Subweb Specifications Language (SWSL) [9], which allows data providers to define how their DKG should be traversed. Inspired by SWSL, traversal-based querying has utilized the Linked Data Platform (LDP) and the Type Index

specification [59]. LDP traversal follows all links within a data space, while Type Index traversal uses mappings from RDF types to relevant resources [69], allowing queries to prioritize implicitly relevant data sources. These contributions are centered on guiding the engine in selecting links to follow in a discovery process. However, they do not explicitly address the restriction or pruning of links after the discovery process based on information acquired during traversal. Such pruning could significantly reduce the query search domain when information about the DKG data model is available. For instance, the structural properties of a subweb could inform the query engine that certain web sections follow a specific data model, allowing a set of IRIs to be pruned from those selected during discovery. In contrast, when LTQP models DKGs as Linked Open Data, the web is not divided into subsections with structural properties; thus, data model information cannot be inferred. To the best of our knowledge, no prior work has explored the use of a pruning mechanism to optimize LTQP, and it is this research gap that the present study aims to address.

RDF Data Shape

RDF data shapes (in this paper, we also refer to them as shapes) are used for validating, describing, and communicating data structures, as well as generating data and driving user interfaces [23, 26]. The two most well known RDF data shape formalisms are SHACL [25] and ShEx [26]. For common use cases, they are equally expressive and interchangeable [24]. RDF data shapes have already been used in the literature for querying centralized KGs [48, 17]. Shape Trees [54] are an index structure for validating and organizing decentralized knowledge graphs (DKGs). However, to the best of our knowledge Shape Trees have not been used for query optimization. Due to their *virtual hierarchy* [54], it can be challenging for a query engine to efficiently capture the relationship between a resource IRI and its corresponding shape. Moreover, Shape Trees are not yet widely adopted; therefore, for the purposes of this work, we use the Shape Index specification [62] to facilitate the mapping between shapes and knowledge graphs. Additionally, automatic generation of RDF data shapes from KGs [22] and shape-based data integration [38] have been studied and can support shape-based summary approaches for DKGs.

Source Selection

Source selection is a crucial challenge in decentralized querying [36, 29]. Link pruning in LTQP is closely related to source selection, as it can be viewed as a dynamic form of source selection. Methods such as basic statistics on triple counts, VoID descriptions, and histogram techniques have been explored in the context of federated querying [36, 29, 42]. However, most of those source selection methods face the limitation of assuming a small number of data sources [29], leaving their suitability for LTQP uncertain. Bloom filters [18] are also a mechanism that has shown success for federated DKGs, yet in the context of LTQP, it has been show that bloom filters have little effect on performance [28]. Schema-based indexing using ontologies [57] has also been explored for source selection of SPARQL queries. It has been shown that this approach is sensitive to the high reuse of vocabulary terms in RDF [29], which is exacerbated in the context of LTQP. The use of implicit RDF schemas for query optimization has been explored through the concept of characteristic sets [43, 41, 42]. However, their applicability to LTQP has not been investigated, and they assume that the entire dataset resides in memory, which is not the case for LTQP.

Preliminaries

RDF Knowledge Graphs and SPARQL Queries

Our work focuses on the union of conjunctive queries over RDF knowledge graphs (KG) using the SPARQL query language [67]. The fundamental building blocks of KGs and SPARQL queries are triples and triple patterns, respectively, as defined in Definition 1 and Definition 2.

Definition 1. Triple. *RDF triple $t = (s, p, o)$ are tuples formed with three terms. A subject where $s \in \mathcal{I} \cup \mathcal{B}$, a predicate $p \in \mathcal{I}$ and an object $o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$. Where $\mathcal{I}$, $\mathcal{B}$, $\mathcal{L}$, are respectively the set of every possible IRI, blank node, literal. For simplicity, we denote the union of these sets by concatenating their symbols, so that $\mathcal{I} \cup \mathcal{B}$ is written as $\mathcal{IB}$.*

Definition 2. Triple pattern. *Triple patterns $tp = (s_{tp}, p_{tp}, o_{tp})$ are similar to triples, where $s_{tp} \in \mathcal{IBV}$, $p_{tp} \in \mathcal{IV}$ and an object term $o_{tp} \in \mathcal{IBVL}$. Where $\mathcal{V}$ is the set of every possible variable.*

We also define two access functions to get respectively the subject and object term of a triple pattern or a triple while ignoring literals, $S : (\mathcal{IBV}, \mathcal{IV}, \mathcal{IBV}) \rightarrow \mathcal{IBV}$ and $O : (\mathcal{IBV}, \mathcal{IV}, \mathcal{IBV}) \rightarrow \mathcal{IBV}$. We denote $[\![Q]\!]^G$ as the evaluation of a query Q over a KG G [3].

Reachability Criteria

LTQP defines completeness on the traversal of links instead of the query results [31]. To formalize the completeness of queries, *Reachability criteria* [31] have been formalized. Reachability criteria are boolean functions (c_i) restricting the dereferencing of links from the internal data source of the query engine. They take as parameters an RDF triple t from an internal triple store, a dereferenceable IRI iri from t, and a union of conjunctive queries Q. If c_i returns $true$, the query engine must dereference iri. More formally

$$c_i(t, iri, Q) \rightarrow \{\text{true}, \text{false}\} \quad (1)$$

Decentralized Knowledge Graphs and Subweb

We define a DKG as a KG G materialized in a network of resources R. A resource $r_i \in R$ is mapped to a KG $g_i \subseteq G$, which is a set of triples [49]. We denote this mapping $r_i \mapsto_{\mathcal{G}} g_i$. A resource is mapped and exposed by an IRI iri_i denoted by $iri_i \mapsto_{\mathcal{R}} r_i$. The network forms a graph where the resources r_i are the nodes and the $iri_j \in g_i$ are directed edges starting from r_i to r_j. G is formed by the union of all the $g \in \text{dom}(R)$. A subweb is a (sub)DKG defined by a set of IRIs controlled by a data provider.

```
@prefix sw: <http://subweb/> .
@prefix si: <https://shapeIndex.com#>.

sw:shapeIndex si:shapeIndexLocation si:
        shapeIndex;
    a si:ShapeIndex;
    si:entry _:user;
    si:entry _:Post.
_:user si:shape sw:user_shape;
    si:subweb "http://subweb/user/{info}".
_:Post si:shape sw:post_shape;
    si:subweb sw:posts.
```

```
PREFIX ex: <http://example.com/>

SELECT * WHERE {
  ?comment ex:creator ex:user;
    ex:content ?content;
    ex:reply ?message.
  ?message ex:hasTopic ?topic;
    ex:forum ?forum.
  ?forum ex:name ?name.
}
```

Figure 2. On the left, an example illustrates a shape index that maps a set of IRIs, represented using a URI template [27], to a user shape, and a specific IRI to a post shape. On the right, an example illustrates of a graph star pattern where the main subject is `?comment` and is linked to the `?message` and `?forum` star patterns.

Approach

This section defines result-based completeness in LTQP, introduces shape indexes, and shows how they enable pruning via a query-shape subsumption algorithm.

Result-Based Completeness in LTQP

Our approach of pruning in LTQP focus on ensuring result completeness, assuming traversal completeness is already defined using reachability criteria. By concentrating on result completeness, we explore strategies to optimize the search space of link traversal queries through pruning of irrelevant resources. We formalize result-based completeness in LTQP as follows. A query is executed over a DKG G formed by the union of all the g in a network R. The query engine has to build a KG G' using a reachability criterion C' in its internal data store from the KGs g by dereferencing resources $iri \mapsto_{\mathcal{R}} r \in R$. We formulate an optimization problem to minimize the size of G', where the query engine constructs a knowledge graph $G'' \subseteq G'$, potentially smaller, by defining a reachability criterion C''. We focus on maintaining the same result completeness, so when using C'' the following equation must hold

$$[\![Q]\!]^{G''} = [\![Q]\!]^{G'} \tag{2}$$

for any network R. Since each $g \in G''$ is obtained by dereferencing resources $r \in R$, a smaller G'' compared to G' implies that fewer HTTP requests were needed to answer the query. Query execution is generally faster with a smaller KG instance, and HTTP requests, being slow and unpredictable [33], can dominate execution time. Therefore, reducing HTTP requests provides a twofold benefit, fewer resources to process and faster query execution.

Shape Index

Pruning in LTQP requires knowledge of the data models of dereferenced resources. However, obtaining complete, up-to-date, and detailed information for each resource in a large decentralized network is impractical. To address this, we introduce the *shape index* as a mapping between RDF document sets and RDF data shapes that describes a subweb controlled by a data provider. Unlike triple statistics, shapes are independent of the KG's size or updates that remain compliant, making them a more cost-effective solution for use cases with stable data models.

We formalize a shape index as follows:

$$SI = \{s_1 \mapsto IRI_1, s_2 \mapsto IRI_2 \cdots, s_n \mapsto IRI_n\} \tag{3}$$

where s_i is a shape and IRI_i is a set of IRI given n entries. The subweb described by the index is defined by $D_{SI} = \bigcup_{IRI \in \text{codomain of } (SI)} IRI$. We denote a shape index as *complete* when every shape $s_i \in \text{dom}(SI)$ has a closed world assumption [25, 26] (we also refer to them as closed) or *incomplete* otherwise. A mapping between a shape and a set of IRIs has implications in the distribution of the data in D_{SI}. When a shape s is mapped to an IRI, then the KG targeted by the mapping $G = \{g \mid \forall iri \in D_{SI}(iri \mapsto_{\mathcal{R}} r \wedge r \mapsto_{\mathcal{G}} g)\}$ satisfies s. Given that the shape is closed, then every set of triples in the resource mapped to an $iri \in D_{SI}$ satisfying the shape must be in a resource mapped to an $iri \in IRI$. We provide a complete description of the shape index in an online specification * and an example of serialization in Figure 2.

RDF data shapes use *targets* to identify the set of nodes or entities in a KG to validate. In this work, we assume all entities in a KG within a document follow the same RDF data shape. We call these entities graph star(s) (patterns), an extension of the RDF star patterns concept [37]. Defined in Definition 3, graph star(s) (patterns) serve two purposes: defining targets for validation and capturing relationships between triple patterns and shape entities. Star patterns consist of triples with the same subject. We extend this concept by linking star patterns where the objects of the triples in one star pattern are the subjects of another, forming a graph structure. For example, a user linking to their posts with recursive replies can be captured with a root star pattern for the user and nested patterns for the posts and replies. Thus, the target of the shapes in the shape index correspond to the subject of each root star pattern when a KG is divided into graph stars with no shared partial graph stars. Figure 2 illustrates an example of a graph star pattern.

Definition 3. Graph Star Pattern (GSP). *We define a star pattern Q_{star} as a set of $tp \in Q$ [37] with the same subject such that given a builder function*

$$BQ_{star}(s) = \{tp_i \in Q \mid S(tp_i) = s\} \tag{4}$$

with $s \in \mathcal{IBV}$ then $Q_{starG} = BQ_{star}(s)$. We define a GSP Q_{starG} as the union between a root star pattern Q_{star_s} and the star patterns having as subject term an object term of another star pattern in Q_{starG}. We define a function $O_{star} : q \in Q \to \mathcal{IBV}$ that returns every non-literal object terms of a star pattern.

We then define Q_{starG} given a set of partial GSP Q_{starGT}

$$Q_{starG} = \bigcup_{q \in Q_{starGT}} q \tag{5}$$

where Q_{starGT} is formed with a root Q_{star_s} by

$$Q_{starGT_i} = \begin{cases} \{Q_{star_s}\} & \text{if } i = 1 \\ \left\{ BQ_{star}(o) \mid o \in \bigcup_{q \in Q_{starGT_{i-1}}} O_{star}(q) \right\} & \text{if } i > 1 \end{cases} \tag{6}$$

*https://constraintautomaton.github.io/shape-index-specification/

We also define a function $S_{star} : q \rightarrow \mathcal{IBV}$ returning the subjects of the Q_{starGT_i} of a Q_{starG}.

We propose a similar definition for the context of KGs where we replace the query Q by a KG G. We denote this structure a graph star.

The construction and maintenance of shape indexes are beyond the scope of this work. Although not evaluated here, shape indexes seem to require less effort to generate than VoID descriptions [10], as they do not include detailed statistics such as triple counts. Nonetheless, VoID descriptions have been successfully employed for query optimization in federated queries [42].

Exposed schemas can also enhance interoperability, which is important in many application domains [51, 8, 7, 50]. Thus, data publishers may have incentives to expose a shape index not only to improve query engine performance but also to satisfy other domain-specific requirements.

RDF data shapes can be prescriptive or descriptive. For descriptive shape indexes, automatic RDF data shape generation methods [22] can facilitate their creation. Entries in shape indexes correspond to sets of IRIs, which can be structured using URI templates [27], reducing the need for exhaustive redefinition. For prescriptive shapes, contributions in shape-based data integration [38] can help prevent the generation of invalid resources.

Overall, shape indexes are lightweight metadata that can be maintained with relatively low effort, especially compared to statistical summaries such as VoID. Publishing them not only benefits query optimization but also contributes to FAIR data principles [68] by enhancing machine-actionability and interoperability. Future work should investigate construction and maintenance strategies for shape indexes in practical deployment scenarios.

Link Pruning Using Shape Indexes

In this section, we establish the connection between shape indexes and link pruning in LTQP as a means to reduce the search domain. In our method, rather than traversing the entire DKG D associated with a shape index, the engine traverses a subgraph $D' \subseteq D$, effectively ignoring resources that are knowably irrelevant to the query. Our approach involves dynamically constructing new reachability criteria during traversal that are more selective as we discover and analyze shape indexes. These criteria are designed so that they will always produce the same completeness of results as the one that was defined at the beginning of the traversal.

To define more selective reachabilities, we propose extending the reachability criteria by formalizing a chain of criteria in a concept called *composite reachability criteria*. In this form, a reachability criterion cp_i is said to *prune* links, and cd_i is said to *discover* links. Equation 7 formalizes a composite reachability criterion C. where Cd is the set of every $cd_i(t, iri, Q)$ and Cp the set of every $cp_i(t, iri, Q)$ used by the engine.

$$C(t, iri, Q) = \bigvee_{cd \in Cd} cd(t, iri, Q) \wedge \bigwedge_{cp \in Cp} cp(t, iri, Q) \tag{7}$$

To perform pruning in LTQP with shape indexes, an initial reachability criterion C_0 is defined. This criterion must include a discovery reachability criterion $cd_{\text{shape index}}$ that leads to a shape index document. After dereferencing a shape index SI_i, the query engine creates a set of links IRI_p containing the links to prune. The links to prune are identified by evaluating the shape index to find IRIs that are not relevant to the query, such that Equation 2 holds, given that G' is produced using C_0. This is done by performing a query-shape subsumption check ($\sqsubseteq_{qs}$), defined in the next section.

We define

$$IRI_p = \left\{ \bigcup SI_i(s_j) | Q \sqsubseteq_{qs} s_j = \text{false} \wedge s_j \in \text{dom}(SI_i) \right\}$$

From this sets of links we define a pruning reachability criteria;

$$cp_{si}(t, iri, Q) = iri \notin IRI_p \tag{8}$$

The new reachability C_i is created by taking the Cd and Cp of C_{i-1} and adding cp_{si} to Cp.

This approach has three main limitations. First, it assumes that data providers maintain up-to-date shape indexes; outdated indexes may lead to incomplete results. A similar criticism could be leverage against the method exploiting VoID descriptions [42]. Second, if the query-shape subsumption check requires dereferencing all documents, it becomes ineffective and may slow down query execution. Third, the approach does not consider cases where querying irrelevant documents could uncover relevant ones via additional reachability criteria. Addressing this would require translating these criteria into queries, which is beyond the scope of this paper.

Query-Shape Subsumption

To determine whether the contents of a resource conforming to a shape is query-relevant, we define a *query-shape subsumption* problem. This problem involves determining whether a GSP in a query can produce at least one result when evaluated over any source conforming to a specific shape, denoted as $Q \sqsubseteq_{qs} S$, meaning Q is subsumed by S. A common approach for validating shapes over an RDF graph is to translate shapes into SPARQL queries [†] [39, 14, 66, 17]. We denote the transformation of a shape S into a query as $T(S)$, which yields a query Q_s. We transform open shapes as queries over the entire KG, since they impose only the minimal constraints required of a KG. When the problem is expressed as GSP $\sqsubseteq_{qs} Q_s$ (considering a GPS to be a `SELECT` query), we say that a GSP is subsumed by S if every result of the GSP can be extended to a result of Q_s, and every triple pattern in a GSP are equivalent to or a specialization of those in the Q_s. Thus, the problem diverges from traditional query containment and query subsumption [55, 47] under set semantics. Query-shape subsumption does not consider only the set of solution mappings but also the constraints of the queries. [‡] The complexity of the problem is reduced by the fact that Q_s has a GSP structure where predicates are always

[†] We only consider shapes that can be transformed into a single `SELECT` SPARQL query [14].

[‡] In this sense, it bears some similarity to query containment under bag semantics, particularly through the notion of "goals-onto" containment mappings [13, 2].

IRIs. This structure arises because shapes primarily describe predicate and object terms and are isomorphic to the specific KG. By exploiting this structure, it is possible to design an algorithm with polynomial-time complexity. Moreover, empirical studies suggest that real-world queries tend to be relatively small [19, 11], making this algorithm practically applicable.

More formally, for a S to subsumed a shape GSP, we consider the queries Q and Q_s, where Q_s is the query translated from S, and both are of the form

$$Q_i = Q_{\text{body}} \bowtie Q_{\text{unions}}$$

Let Q_{body} denote the Basic Graph Pattern (BGP) of the query, and let $Q_{\text{unions}} = \bigcup Q_u$ represent the Union Graph Patterns (UGPs) [67] expressed in normal form. Here, each Q_u is of the form $q_0 \cup q_1$, where each q_i is a BGP, and we assume that the q_i contain no union statements themselves.

Algorithm 1 Check if a GSP is subsumed by Q_s ($subsums_{\text{graphstar}}$)

Input: $Q_{star}, Q_{starG_i}, Q_s = Q_{\text{sbody}} \cup Q_{\text{sunions}}, Eval_{star}$
Output: **true** or **false** whether the root of a graph star pattern Q_{star} subsumes a shape.

```
1:  if M(S_star(Q_star)) ∈ Eval_star then
2:      return M(S_star(Q_star))
3:  end if
4:  for all tp ∈ Q_star do
5:      if not match(tp, Q_sbody) then
6:          hasOnePath ← false
7:          for all q_us ∈ Q_sunions do
8:              if subsums_graphstar(Q_star, Q_starG_i, q_us, Eval_star) then
9:                  hasOnePath ← true
10:             end if
11:         end for
12:         if not hasOnePath then
13:             Eval_star ← Eval_star ∪ (M(S_star(Q_star)) ↦ false)
14:             return false
15:         end if
16:     else
17:         if O(tp) ∈ S_star(Q_starG_i) then
18:             Eval_star ← Eval_star ∪ (M(S_star(Q_star)) ↦ true)
19:             if not subsums_graphstar(Q_star_O(tp) ∈ Q_starG_i, Q_starG_i, Q_s, Eval_star) then
20:                 Eval_star ← Eval_star ∪ (M(S_star(Q_star)) ↦ false)
21:                 return false
22:             end if
23:         end if
24:     end if
25: end for
26: return true
```

Algorithm We define the function $subsums_{\text{graph star}}$ in Algorithm 1 to evaluate whether a GSP with a root star pattern Q_{star_i} from Q_{starG_i} is sumbsumed by Q_s. The algorithm also takes a set $Eval_{star}$ to track which partial graph star patterns have already been evaluated. The algorithm iterates over each triple pattern in the root star pattern Q_{star_i} and uses the $match$ function to check whether there exists a triple pattern in the BGP of Q_s whose domain of matched triples is a superset of that of the current pattern. If the triple pattern cannot be found in the BGP, the algorithm then looks into the UGPs of Q_s. Since we assume that the union statements are not nested, this limits the number of recursive calls. If an equivalent triple pattern is found, the algorithm checks whether the object of the triple pattern is the subject of a partial graph star pattern in Q_{starG_i}. If it is, the algorithm recursively applies the same procedure to this partial GSP. To avoid cycles and redundant evaluations we maintain a set of evaluated answer in $Eval_{star}$. We notice that the complexity of Algorithm 1 is $O(n_{tp}^2 \cdot n_{sunion})$ where n_{tp} is the number of triple patterns of Q_{starG_i} and n_{sunion} the number of UGP in Q_s. To solve $Q \sqsubseteq_{gs} S$, we need to consider the number of GSP from the BGP with their number of segments in the UGP and the number of BGPs in the UGPs. This operation results again in a polynomial time complexity algorithm. The following paragraphs analyze the time complexity of the algorithm.

Time Complexity Analysis

Worst-case per Q_{star} Let n_{qstar} denote the number of distinct $Q_{star} \in Q_{starG_i}$. For each node Q_{star}, the algorithm iterates over its triple patterns (line 4 to 25), resulting in a time complexity of $O(n_{tpQ_{star}})$. For each triple pattern that does not match the shape body (Q_{sbody}), the algorithm iterates over all union branches in Q_{sunion} (line 7 to 11), making at most n_{sunion} recursive calls. The algorithm traverse the q_{us} graph however each $q_{us} \in Q_{sunion}$ cannot contain a Union Graph Pattern (UGP), and is therefore always of the form $Q_s = Q_{\text{sbody}}$, making this branch (line 5 to 15) of the algorithm after a first execution not the worst case with a complexity of $O(n^2_{tpQ_{star}} \cdot n_{sunion})$.

After the first execution, the worst-case scenario becomes one in which the triple patterns matches a pattern in Q_{sbody}, and the condition $O(tp) \in S_{star}(Q_{starG_i})$ holds (line 17 to 22). In such cases, the algorithm recursively explores the corresponding partial Graph Star Pattern (GSP) by executing $subsums_{\text{graphstar}}$, following a graph traversal paradigm.

Tree Traversal Argument Although Q_{starG_i} is a graph, the algorithm avoids cycles due to "caching", once a node Q_{star} is evaluated, its result is stored in $Eval_{star}$. Therefore, each node is visited at most once, and the overall traversal is equivalent to a tree traversal of size $n_{qstar} \leq |Q_{starG_i}|$.

Total Complexity Let $n_{tpQ_{star}}$ denote the number of triple patterns in a particular Q_{star}, and let n_{tp} be the total number of triple patterns across all nodes, so

$$n_{tp} = \sum_{Q_{star} \in Q_{starG_i}} n_{tpQ_{star}}$$

Then the total number of recursive operations over all nodes is bounded by:

$$O\left(\sum_{i=1}^{n_{qstar}} n_{sunion} \cdot n^2_{tpQ_{star_i}}\right)$$

$$O\left(n_{sunion} \cdot \sum_{i=1}^{n_{qstar}} n^2_{tpQ_{star_i}}\right)$$

$$O\left(n_{sunion} \cdot \left(n^2_{tp} - \sum_{i=1,j=1}^{n_{qstar}} n_{tpQ_{star_i}} \cdot n_{tpQ_{star_j}}\right)\right)$$

$$O\left(n_{sunion} \cdot n^2_{tp}\right)$$

Time Complexity of the $subsums_{\text{Q}}$ Algorithm The time complexity of the $subsums_{\text{Q}}$ algorithm is straightforward to derive, as it consists of iterating over the Graph Pattern Structures (GPS) of the query and applying the $subsums_{\text{graphstar}}$ algorithm. Thus, the time complexity is

Algorithm 2 Check if a query Q subsumes a Q_s ($subsums_Q$)

Input: Q, Q_s and $Eval_{star}$
Output: **true** or **false** whether the root of a graph star pattern Q subsumes a shape.

```
for all Q_starG_s ∈ Q do
    for all Q_star_s ∈ Q_starG_s do
        if subsums_graphstar(Q_star_s, Q_starG_s, ∅) then
            return true
        end if
    end for
end for
return false
```

given by:

$$O\left(n_{sunion} \cdot n_{tp_{max}}^2 \cdot n_{Q_{star_{max}}} \cdot n_{Q_{starG}}\right) \quad (9)$$

Where, $n_{tp_{max}}$ is the maximum number of triple patterns in any Q_{starG}, $n_{Q_{star_{max}}}$ is the maximum number of Q_{star} patterns in any Q_{starG}, $n_{Q_{starG}}$ is the number of Graph Pattern Structures (GPS) in the query Q.[§]

Experimental Setup

We implemented our approach using the LTQP version of the Comunica query engine [61]. We chose Comunica due to its modularity [60] and its established use in several LTQP studies [9, 59, 20, 28, 21, 62]. All implementations are open-source and are provided in the supplementary material. Similarly to other LTPQ studies, we used SolidBench [59], which is based on the LDBC social network benchmark [4], to evaluate our contribution. Furthermore, SolidBench was developed because no LTQP benchmark existed prior to its creation [32, 59], and, to the best of our knowledge, no other benchmarks have been introduced since. We created an open-source module to generate shape indexes in SolidBench, based on user-provided mappings between ShEx shapes and data model objects. The shape-annotated portion of the data model includes posts, comments on posts, user profiles, cities, and likes. The datasets are Solid Pods [52, 16]. In this paper, we consider a Solid Pod as a web-based file system that follows the LDP specification [56]. Each Solid Pod, contains alongside its data, a shape index and separate resources for each shape definition. Some shapes are nested within others. For example, user profiles are associated with cities, and comments are associated with posts. Depending on the pod instance, certain data model objects are materialized in a single file, while others are distributed across multiple files. The benchmark provides queries that simulate typical read actions in social media use cases, such as retrieving replies to posts or identifying users connected to a given user. Some queries are highly selective, while others return larger result sets. Additionally, some queries request many data model elements (e.g., profiles, comments, cities), whereas others request only a few. The datasets contained approximately 4,200,000 triples and 1,528 subwebs, which, in this context, are Solid Pods. The shape indexes contain 13 triples each, while the largest shapes have up to 150 triples. This can be considered insignificant, particularly because the number of triples does not scale with the size of the subwebs. The entire data model and query templates are available in the supplementary material.

To evaluate our approach, we conducted the following experiments: we first measured the execution time and results of our query-shape subsumption algorithm using the shapes from the study; We then compared our shape index approach to state-of-the-art Solid Pod network traversal algorithms: one leveraging the type index specification [59], and another using the LDP specification [59], in a network where each Solid Pod provides a complete shape index. It should be noted that the type index and shape index approach are extensions of the LDP approach. Subsequently, we analyze the relationship between the number of HTTP requests and query execution time, drawing on data collected across all experiments. Additionally, we assessed the resilience of our approach by gradually reducing the shape index information across the network. We measured query execution time in networks where 0%, 20%, 50%, and 80% of Solid Pods expose a shape index. We also varied the percentage of shape index entries using closed shapes, testing 20%, 50%, and 80%. We finally evaluate a network having shapes that incorporate only data from the Solid Pods and shapes providing a minimal dataset description where the object constraints are always an IRI or a literal.

We conducted the experiment using queries from five different instantiations of SolidBench query templates, varying the starting pods in a random yet reproducible manner. Experiments were repeated 50 times with a 2 minute timeout per query execution. They were conducted on an Ubuntu 20.04.6 LTS machine with a 2x Hexacore Intel E5645 CPU and 24GB RAM.

Discussion of Results

Evaluation Against Other Approaches

Figure 3 shows that the shape index approach performs better or comparably to the state-of-the-art Solid Pod network traversal algorithms, for all query templates except S4. Figure 12 in the Appendix present the execution time of each method and Table 2 present the statistical significance by query templates. The shape index approach allows for answering queries from the S7 template, which was not possible by the other approaches, due to large amount of HTTP requests resulting in timeouts. Queries can require as little as 13% of the execution time (S1 being 7 times faster) of the type index. The queries that perform the best are those in which the number of HTTP requests decreased the most, as can be inferred from the analysis of Table 1. Table 1 presents the average percentage of query-relevant resources per query template, derived from the where-provenance [12] of the query results and the number of HTTP requests. Queries from templates D6 and D7 show no reduction because they require nearly every document in the dataset to be processed by the engine, making our approach ineffective in these cases. We notice that queries from template S4 with the shape index performed worse in every instance, with an increase in query execution time of up to 2.80 times. This is further illustrated in Table 1, which shows that for these queries, the type index traversal algorithm achieves a ratio of query-relevant resources dereferenced of 100% or 50%, compared to only 6% with the shape

[§]Sharing the evaluation results $Eval_{star}$ from $subsums_{\text{graphstar}}$ could reduce execution time and potentially the algorithm's complexity.

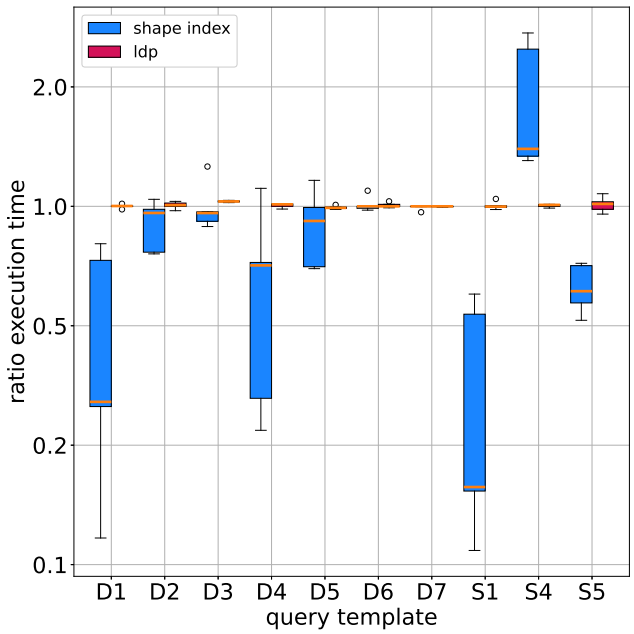

Figure 3. Comparison of query execution time with the type index approach. The ratio represents how the execution time of each method compares to that of the type index. A ratio greater than 1 indicates a slower execution time (**lower is better**). The shape index approach performs similarly or better than the other methods, except in the case of S4.

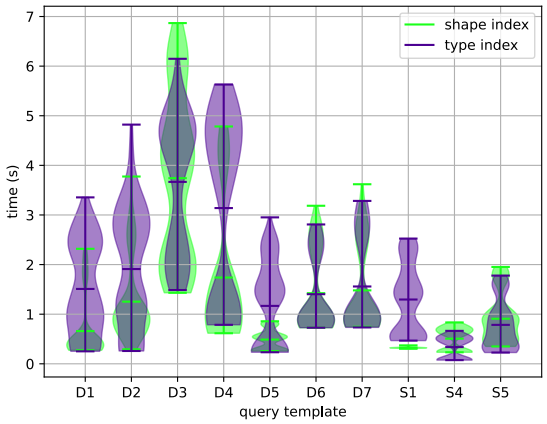

Figure 4. The shape index approach tends to produce the first results earlier than the type index approach, particularly for queries where the shape index achieves shorter total execution times.

index approach. The poor performance is due to the fact that the links acquired by the other approaches were selected based on reachability criteria that did not leverage the structural properties of the dataset, such as in the case of `Cmatch` [33], a reachability criterion based on the structure of the query. In contrast, the shape index approach always enforces the use of these properties, resulting in additional HTTP requests and increased processing time. However, those queries were already efficient, with the type index traversal approach completing in only about 0.30% of the maximum allowed execution time. Nonetheless, these results still highlight a category of queries and networks for which our approach is not well-suited. Those results mostly validate **H1** however the shape index approach can drastically increase the execution time when the structural properties are not used.

Approach/ Query Template	D1	D2	D3	D4	D5	D6	D7	S1	S4	S5
shape index (%)	14^{42}_{6}	38^{91}_{11}	52^{92}_{37}	15^{20}_{3}	9^{13}_{2}	18^{41}_{7}	8^{11}_{2}	12^{12}_{12}	6^{6}_{6}	10^{11}_{4}
type index (%)	4^{6}_{4}	35^{86}_{8}	51^{91}_{36}	9^{15}_{3}	8^{12}_{2}	20^{42}_{8}	9^{13}_{2}	4^{8}_{1}	80^{100}_{50}	3^{4}_{2}

Table 1. For most queries, the percentage of query-relevant resources is low (cell values are shown as $\text{avg}^{\text{max}}_{\text{min}}$). Shape index generally matches or outperforms the type index, except for templates D6, D7, and S4. The difference for S4 is more pronounced, reaching 100% with the Type Index compared to only 6% with the shape index.

To further our analysis, we compare three temporal metrics: the arrival time of the first results, the termination time, defined as the duration between the arrival of the last result and the end of query execution, and the waiting time, which we define as the accumulated time gaps exceeding one second between the reception of consecutive results. We choose a 1 second threshold for waiting time based on research in responsive system design. These waiting time guidelines are derived from the field of neuropsychology and are considered stable regardless of technological expectation for human users [46, 44]. A waiting time below 0.1 seconds is perceived as instantaneous, a delay of up to 1 second preserves a seamless flow of thought, while delays exceeding 10 seconds tend to cause user attention to drift [44]. We additionally computed the diefficiency metric in relation to time (*dief@t*) [1], at the previously mentioned time intervals: 0.1 s, 1 s, and the arrival time of the final result. We did not choose 10 s as the threshold, because queries that terminate complete in under 10 s. Tables 3 and 4 in the Appendix present the average results for the metrics presented above.

Figure 4 presents the time of arrival of the first results for all query instances, grouped by query templates. The plot indicates that, for most queries, the shape index approach tends to produce results faster, as shown by distributions concentrated around lower first result arrival times. Templates D1, D2, D5, and S1 exhibit the most significant improvements in first-result latency when using the shape index. Surprisingly, although queries from templates D2 and D5 sometimes performed worse with the shape index approach, as shown in Figure 3, they still produced the first result faster than with the type index approach. This suggests that, although the shape index does not explicitly prioritize early result generation, the pruning it performs may implicitly favor faster first results arrival with some queries and networks. This implicit prioritization does not apply uniformly across all templates. For example, queries from template D3 exhibit faster execution times across most instances when using the shape index. However, as shown in Figure 4, the shape index also results in a higher mean and a longer tail for the time to first result. This difference arises from the structure of the subwebs targeted by D3 queries. In some cases, a result can be obtained from a document located one link away from the seed document. The direct dereferencing using the type index approach can exploit this proximity, enabling early result generation. In contrast, the shape index approach may delay first result production because it relies on discovering the shape index documents before exploring the rest of the subweb.

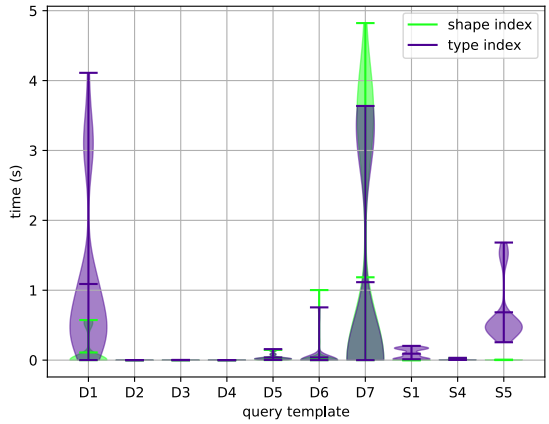

Figure 5. With the exception of queries from templates D6 and D7, which have similar execution times, the shape index either maintains or improves query termination time.

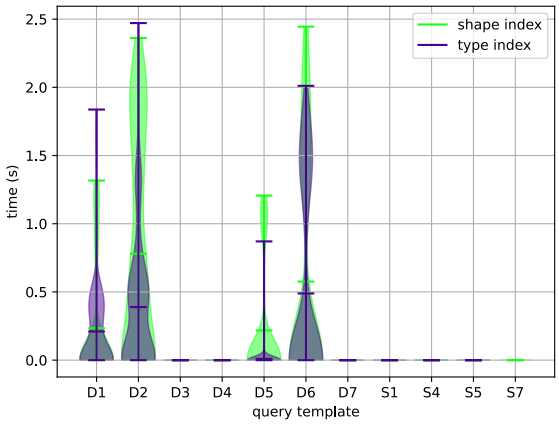

Figure 6. Except for queries from template D1, the shape index approach tends to either maintain or worsen the waiting time.

Figure 5 presents the termination times for the query execution instances, grouped by template. For queries where execution time improved and termination time was non-zero, such as those from templates D1, S1, and S5, the shape index approach significantly reduces termination time. This improvement aligns with the substantial increase in the percentage of query-relevant resources dereferenced when using the shape index, as shown in Table 1. A long termination time generally indicates that, toward the end of execution, a large number of non-query-relevant resources are being dereferenced. Although Table 1 does not show the temporal progression of this percentage it still give us information about the quantity of non-query-relevant resources being dereferenced. For templates D5 and D6, the termination-time distributions for the shape index approach have a similar shape but exhibit a longer tail, particularly D6, which can take up to one second longer to terminate. In these cases, Table 1 shows that the proportion of query-relevant resources dereferenced using the shape index is within $\pm 1\%$ of that obtained with the type index. A similar situation is observed for template D3, although in this case the type index achieves a termination time of zero. The longer termination times for D5 and D6 may be attributed to the additional dereferencing of shape index documents. Notably, for template D6 in Figure 3, an outlier query increases execution time by approximately a ration of 1.2, which could explain the extended tail and the additional second of termination time.

Figure 6 presents the waiting time for the query instances, grouped by template. With the exception of queries from the D1 template, which show a decrease in waiting time, the shape index approach generally increases the waiting time for queries that experience a delay before producing results. This increase is likely due to the additional dereferencing of shape index documents that must occur before relevant results can be generated. Such dereferencing can introduce a gap before the first results appear, even when the total query execution time is reduced or remains unchanged.

Figures 7, 8, and 9 show the *dief@t* values measured at 0.1 seconds (*dief@0.1s*), 1 second (*dief@1s*), and at the time of the last result (*dief@lr*).[¶] The *dief@0.1s* results

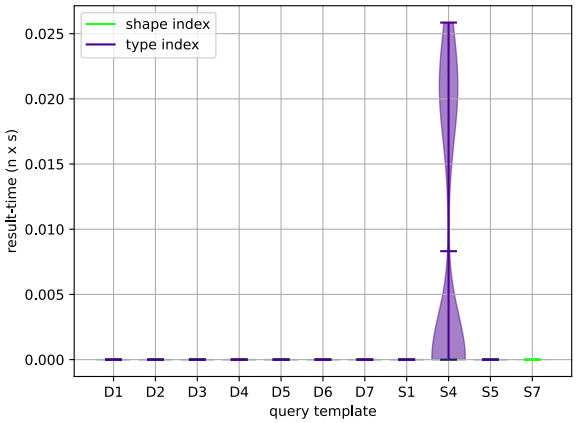

Figure 7. At 0.1 second, the value is zero for every query template except S4. This is expected, as for all other templates the first result arrives after 0.1 seconds.

show that, except for S4, no approach is able to produce instantaneous results, which aligns with the observations in Figure 4. Overall, this indicates that, with the current approaches and benchmark, producing instantaneous results is not feasible. The *dief@1s* results indicate that the shape index approach generally produces a higher number of early results, except for queries from templates S4 and S5. This result is expected for S4 as the performance with the shape index is drastically reduced as shown in Figure 3 however with S5 it is less expected as it is a query template that perform better with the shape index. It would be expected to see a performance improvement in terms of *dief@t* at the last results for S5, however, the values are zero or close to zero. This is because these queries produce only a single result, so *dief@t* naturally tends toward 0 unless there is a very large difference in execution time between the two approaches (see footnote ¶). These results suggest that, for fast early result arrival, the shape index approach tend to performs better. This finding is particularly relevant in the context of

[¶]This means that if, for a given query, the shape index approach produces the last result at 5 seconds and the type index at 5.10 seconds, then the *dief@t* metric will be computed at 5.10 seconds.

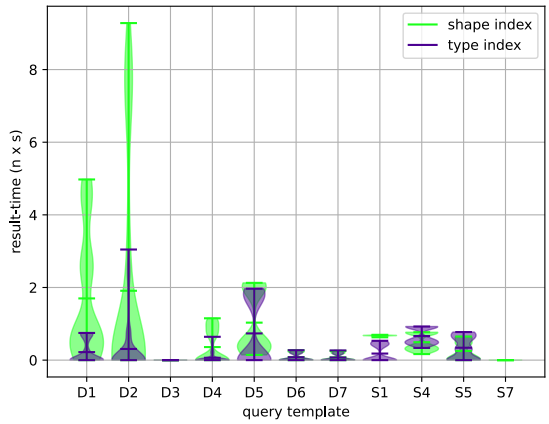

Figure 8. At 1 second, the shape index approach shows a higher *dief@1s* (**higher is better**), indicating that more results are obtained while maintaining a seamless flow of thought.

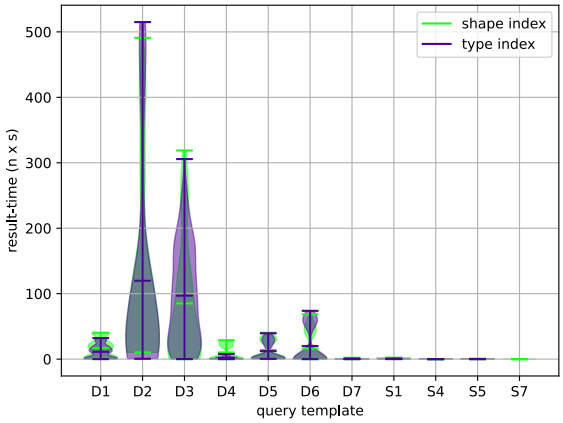

Figure 9. At the time of the last result, the shape index approach tends to exhibit a similar *dief@t* (**higher is better**).

social media applications, where it is often more important to obtain fast, relevant results to maintain interactivity than to retrieve complete results. The *dief@lr* show a similar behavior but with lesser performance gain compare to at *dief@1s* (see Table 3 to compare the average values).

Query-Shape Subsumption Evaluation

The empirical evaluation of the query-shape subsumption algorithm shows that its execution time with the more detailed shapes from our experiment is negligible, with a maximum execution time of 4.655 ms (0.0039% of the timeout). Table 5 present does results. This outcome is expected, as the algorithm has polynomial time complexity, and the shapes and queries in the experiments are small and not deeply nested. This result validated **H2**.

Evaluation of the Resilience of the approach

The final part of the results analysis focuses on the resilience of the shape index approach. In this analysis, we examine the impact of reducing the shape index information in the network and compare the results with a network in which all pods are exposed to detailed, complete shape indexes.

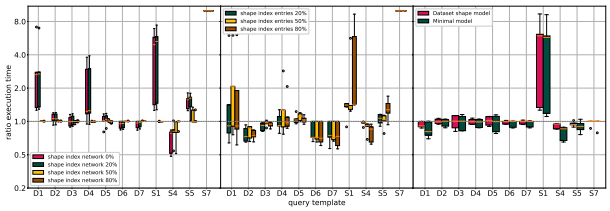

Figure 10. Shape index approaches tend to perform less effectively with limited network information and comparatively better where the baseline shape index underperforms. A higher ratio indicates a longer query execution time compared to a network with complete shape index information (**lower is better**).

Figure 10 presents three plots that illustrate the results of our evaluation of the approach's resilience. The plot on the left shows the variation in the availability of shape indexes across the network. As expected, we observe that queries that performed better in Figure 3 tend to perform worse with reduced shape index information, while queries that performed poorly improved.

The plot in the middle shows the variation in the percentage of shape index entries using closed shapes. The results here are more nuanced. While there is a general trend for query evaluations with a lower percentage of closed shapes to behave similarly to the plot on the left, we also observe both performance gains for some query template and a drastic performance loss for the queries of S1 when 80% of the shape entries are closed. The performance gain occurs because not every entry needs to be closed to prune the query irrelevant documents. Entries mapped to an open shape are always considered relevant because the shape translates to a query fetching the whole KG. If the subsumption check leads to the same conclusion, then, given a closed entry, the execution will be more expensive because when shapes are nested, the nested shapes need to be dereferenced to solve the query-shape subsumption algorithm. For the queries of S1, with 80% of closed shape entries, the performance lost was due to random chance, as the discriminatory entries were provided with open shapes in multiple instances when looking at the raw data.

The right plot shows the variation in the level of detail of the shapes by reducing their detail. Since the shapes are closed in this experiment the level of detail was varied by changing the constraint of the object terms. Most queries tend to perform similarly or better, with the exception of those of S1. Upon analyzing the output of our query subsumption algorithm, we observe that the additional information to the shape provided in our base approach does not affect the algorithm's results. However, in the baseline shape index experiment, the engine must dereference more shapes, potentially increasing execution time. Queries of template S1 are the only ones where the added information can discriminate multiple parts of the datasets' domain leading to worse performane. This indicates that in some situations, adding more information can be beneficial. This sensitivity of the quantity of the information in the index also helps explaining the results for S1 queries in the middle plot. In that case, the engine still had to dereference shapes from each dataset, and the information available was likely insufficient to significantly discriminate between resources.

These results show that **H3** and **H4** are rejected: reducing information is not always detrimental to query execution. It becomes detrimental only when the omitted information is critical for determining the query relevance of resources. Therefore, the outcome is dependent on both the query and the network. Moreover, the completeness of shape indexes has a smaller impact on performance than their presence or absence. **H5** is accepted; it is possible to retain performance gains even in networks with reduced shape-index information.

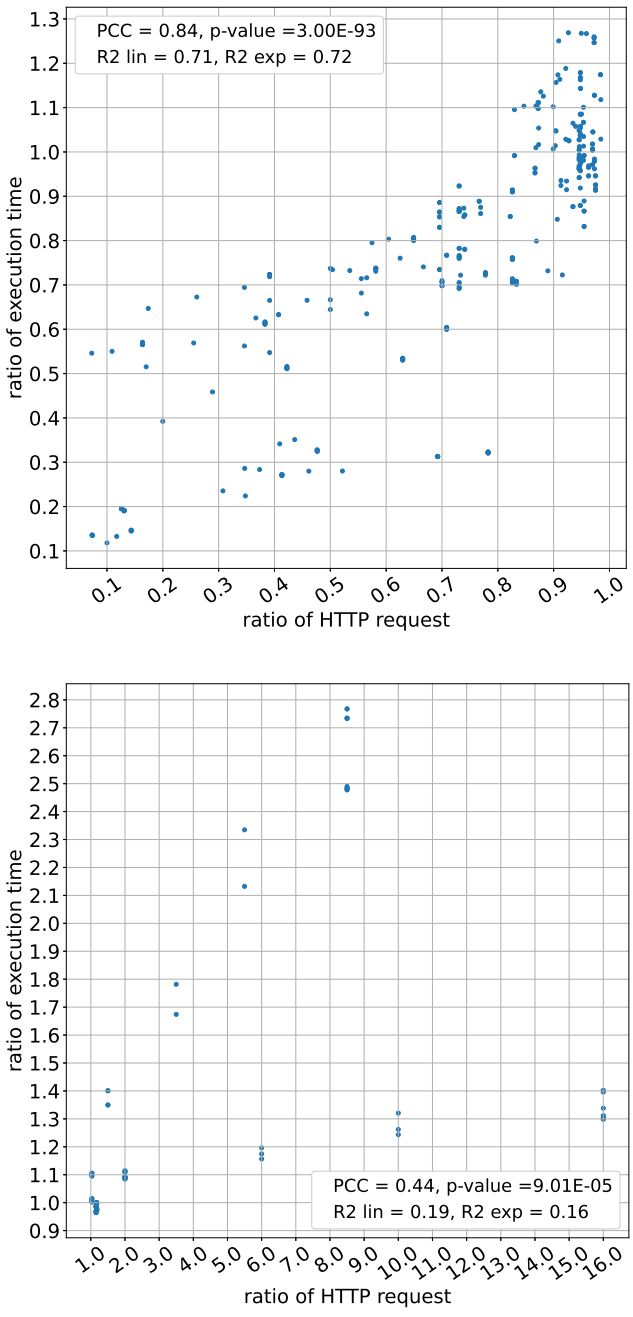

Figure 11. The data show two regimes in the relation between the number of HTTP requests and the execution time, we see a more linear correlation on the up figure than on the down figure.

Relationship Between HTTP Request and Query Execution Time

To study the relationship between the number of HTTP request and the query execution time, the ratio of HTTP request of the different approaches and the ratio of query execution time was calculated in relation to the type index approach. Figure 11 present our analysis. The relationship between HTTP request and query execution time can be divided into two regimes. In the first regime (up figure), where the shape index approach reduces the number of HTTP requests, we notice a positive linear correlation with a Pearson correlation coefficient (PCC) of 0.84 and a high statistical significance (< 0.01). We can notice that toward the end, the curve appears to exhibit a more exponential behavior. Evaluating an R^2 score with an exponential best fit curve we get a score of 0.72 and 0.71 for a linear curve. Above a ratio of approximately 0.85 of HTTP requests, the shape index approach did not guarantee a reduction in query execution time. There can be multiple explanation for this behavior. First, the method have some overhead due to the query-shape subsumption algorithm and the state retention of the pruning reachability criteria. However, as shown in in section the query-shape subsumption algorithm execution time is negligeable for one execution and the state retentation has a polynomial time complexity thus it should not have a high impact on the execution time. Another explanation is the number of HTTP request that are performed in parallel. The LTQP version of Comunica performs 10 HTTP requests in parallel, thus, we would expect that with a low number and ratio of HTTP requests, the performance would remain largely unchanged or slightly worse, which is what can be observed in the curve.

In the second regime (down figure), the shape index increases the number of HTTP requests. We notice a moderate positive linear correlation with a PCC of 0.44. The overall correlation between reducing HTTP requests and query execution time is positively linear, with a moderate linear correlation with a PCC of 0.56 and a high statistical significance. The correlation is more linear than exponential with R^2 scores respectively of 0.31 and 0.24, however due to the low score it is difficult to determine the nature of the distribution.

Explaining the two behavioral regimes observed in the data is challenging. One possible explanation for the poorer performance of the shape index approach in certain cases could be the low number of samples. However, we also observe that the relationship between the two variables differs significantly across the regimes. In the first regime, the relationship is close to one-to-one, with a slope of approximately 0.91. In contrast, the second regime shows a much flatter slope of around 0.08, suggesting that the ratio of HTTP requests has a weaker influence. This disparity indicates that the explanation may be more complex than just a sample size issue. The increased number of HTTP requests can be attributed to queries executed using the D6, D7, and S4 templates, with the S4 template causing the largest increase. Queries from the S4 template typically require only 1 or 2 HTTP requests, so when running with 10 concurrent requests, the impact on performance is limited. On the other hand, queries using the D6 and D7 templates, executed with

the type index traversal algorithm, required 26, 23, and 129 HTTP requests. This significantly higher number of HTTP requests explains why the increase in HTTP requests had a greater impact on query execution time.

These results partially validate **H6**, as a linear relationship is observed when there are a sufficient number of HTTP requests and a significant reduction ratio, taking concurrent requests into account.

Conclusion

In this article, we introduced a pruning mechanism for LTQP, extending the concept of reachability criteria and leveraging shape indexes. Using the SolidBench benchmark, we demonstrate that our approach significantly reduces the number of HTTP requests without degrading performance, particularly when queries exploit the structural properties of the dataset. In the best-case scenario, query execution time improved by up to 7 times, and a previously unexecutable query completed successfully. On average, performance increased by 1.76 times, although in the worst case, execution time rose by 2.8 times for a query that did not exploited the structure of the network. Our method also handles scenarios with partial or reduced shape index information in networks, making it relevant for decentralization efforts that allow third-party clients to efficiently query large, heterogeneous datasets. The approach imposes minimal overhead on servers, requiring only the serving of small static shape index files. This work opens a new line of research in integrating data models directly into decentralized query execution, particularly for unindexed networks. Future work could focus on enhancing query planning in LTQP [58] with RDF data shapes, maintaining and developing shape indexes, and exploring other low-cost, low-maintenance indexing structures.

Supplemental Material Statement: All source code, benchmark queries, datasets, raw results, and supplementary analyses are available online. ‖

References

[1] 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*. Ed. by Claudia d’Amato et al. Cham: Springer International Publishing, 2017, pp. 3–19. ISBN: 978-3-319-68204-4.

[2] Foto N. Afrati, Matthew Damigos, and Manolis Gergatsoulis. “Query containment under bag and bag-set semantics”. In: *Information Processing Letters* 110.10 (Apr. 2010), pp. 360–369. DOI: 10.1016/j.ipl.2010.02.017. URL: https://doi.org/10.1016/j.ipl.2010.02.017.

[3] Renzo Angles and Claudio Gutierrez. “The Expressive Power of SPARQL”. In: *The Semantic Web - ISWC 2008*. Ed. by Amit Sheth et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 114–129. ISBN: 978-3-540-88564-1.

[4] Renzo Angles et al. “The LDBC Social Network Benchmark”. In: *CoRR* abs/2001.02299 (2020). arXiv: 2001.02299. URL: http://arxiv.org/abs/2001.02299.

[5] Amr Azzam et al. “SMART-KG: Hybrid shipping for SPARQL querying on the web”. In: *Proceedings of the Web Conference 2020*. 2020, pp. 984–994.

[6] Amr Azzam et al. “WiseKG: Balanced access to web knowledge graphs”. In: *Proceedings of the Web Conference 2021*. 2021, pp. 1422–1434.

[7] Ceri Binding and Douglas Tudhope. “Improving interoperability using vocabulary linked data”. In: *International Journal on Digital Libraries* 17.1 (2016), pp. 5–21.

[8] Justin Bingham, Eric Prud’hommeaux, and elf Pavlik. *Solid Application Interoperability — solid.github.io*. https://solid.github.io/data-interoperability-panel/specification/. [Accessed 15-08-2025]. 2025.

[9] Bart Bogaerts et al. “Link Traversal with Distributed Subweb Specifications”. In: *RuleML+RR*. 2021. URL: https://api.semanticscholar.org/CorpusID:244848944.

[10] Christoph Böhm, Johannes Lorey, and Felix Naumann. “Creating voiD descriptions for Web-scale data”. In: *Journal of Web Semantics* 9.3 (2011). Semantic Web Dynamics Semantic Web Challenge, 2010, pp. 339–345. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2011.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1570826811000370.

[11] Angela Bonifati, Wim Martens, and Thomas Timm. “An analytical study of large SPARQL query logs”. In: *The VLDB Journal* 29.2–3 (Aug. 2019), pp. 655–679. ISSN: 0949-877X. DOI: 10.1007/s00778-019-00558-9. URL: http://dx.doi.org/10.1007/s00778-019-00558-9.

[12] 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*. Springer. 2001, pp. 316–330.

[13] Surajit Chaudhuri and Moshe Y. Vardi. “Optimization of Real Conjunctive Queries”. In: *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. PODS ’93. Washington, D.C., USA: Association for Computing Machinery, 1993, pp. 59–70. ISBN: 0897915933. DOI: 10.1145/153850.153856. URL: https://doi.org/10.1145/153850.153856.

‖ https://github.com/shapeIndexComunicaExperiment/documentation

[14] Julien Corman et al. "Validating Shacl Constraints over a Sparql Endpoint". In: *The Semantic Web – ISWC 2019*. Ed. by Chiara Ghidini et al. Cham: Springer International Publishing, 2019, pp. 145–163. ISBN: 978-3-030-30793-6.

[15] James Curran. "The internet of dreams Reinterpreting the internet". In: *Misunderstanding the Internet*. Routledge 2012, 2016. Chap. 1.

[16] Ruben Dedecker et al. "What's in a Pod? A knowledge graph interpretation for the Solid ecosystem". In: *6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs (QuWeDa) at ISWC 2022*. Vol. 3279. CEUR. 2022, pp. 81–96.

[17] Delva, Thomas and Dimou, Anastasia and Jakubowksi, Maxime and Van den Bussche, Jan. "Data provenance for SHACL". eng. In: *Proceedings 26th International Conference on Extending Database Technology (EDBT 2023)*. Vol. 26. Ioannina, Greece, 2023, 285–297. ISBN: 9783893180936. URL: %7Bhttp://doi.org/10.48786/edbt.2023.23%7D.

[18] Amadou Fall Dia et al. "Fast SPARQL join processing between distributed streams and stored RDF graphs using bloom filters". In: *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. IEEE. 2018, pp. 1–12.

[19] AnHai Doan, Alon Halevy, and Zachary Ives. "2 - Manipulating Query Expressions". In: *Principles of Data Integration*. Ed. by AnHai Doan, Alon Halevy, and Zachary Ives. Boston: Morgan Kaufmann, 2012, pp. 21–63. ISBN: 978-0-12-416044-6. DOI: https://doi.org/10.1016/B978-0-12-416044-6.00002-8. URL: https://www.sciencedirect.com/science/article/pii/B9780124160446000028.

[20] Ruben Eschauzier, Ruben Taelman, and Ruben Verborgh. "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*. Nov. 2023. URL: https://rubeneschauzier.github.io/QuWeDa2023-Link-Queue-Analysis-Camera-Ready/QuWeDa_2023_Link_Queue_Analysis_Final.pdf.

[21] 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*. Sept. 2024. URL: https://rubeneschauzier.github.io/traversal-metric-workshop-paper/.

[22] Daniel Fernández-Álvarez et al. "Extracting shapes from large RDF data collections". In: (2023).

[23] Jose Emilio Labra Gayo et al. "Applications". In: *Validating RDF Data*. Cham: Springer International Publishing, 2018, pp. 195–231. ISBN: 978-3-031-79478-0. DOI: 10.1007/978-3-031-79478-0_6. URL: https://doi.org/10.1007/978-3-031-79478-0_6.

[24] Jose Emilio Labra Gayo et al. "Comparing ShEx and SHACL". In: *Validating RDF Data*. Cham: Springer International Publishing, 2018, pp. 233–266. ISBN: 978-3-031-79478-0. DOI: 10.1007/978-3-031-79478-0_7. URL: https://doi.org/10.1007/978-3-031-79478-0_7.

[25] Jose Emilio Labra Gayo et al. "SHACL". In: *Validating RDF Data*. Cham: Springer International Publishing, 2018, pp. 119–194. ISBN: 978-3-031-79478-0. DOI: 10.1007/978-3-031-79478-0_5. URL: https://doi.org/10.1007/978-3-031-79478-0_5.

[26] Jose Emilio Labra Gayo et al. "Shape Expressions". In: *Validating RDF Data*. Cham: Springer International Publishing, 2018, pp. 55–117. ISBN: 978-3-031-79478-0. DOI: 10.1007/978-3-031-79478-0_4. URL: https://doi.org/10.1007/978-3-031-79478-0_4.

[27] J. Gregorio et al. *RFC 6570: URI Template — datatracker.ietf.org*. https://datatracker.ietf.org/doc/html/rfc6570. [Accessed 13-05-2025]. 2012.

[28] Hanski, Jonni and Taelman, Ruben and Verborgh, Ruben. "Observations on bloom filters for traversal-based query execution over solid pods". und. In: *ESWC2024, the European Semantic Web Conference*. Hersonissos, Greece, 2024, p. 5.

[29] Andreas Harth et al. "Data Summaries for On-Demand Queries over Linked Data". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 411–420. ISBN: 9781605587998. DOI: 10.1145/1772690.1772733. URL: https://doi.org/10.1145/1772690.1772733.

[30] Olaf Hartig and Carlos Buil-Aranda. "brTPF: Bindings-Restricted Triple Pattern Fragments (Extended Preprint)". In: *CoRR* abs/1608.08148 (2016). arXiv: 1608.08148. URL: http://arxiv.org/abs/1608.08148.

[31] Olaf Hartig and Johann-Christoph Freytag. "Foundations of Traversal Based Query Execution over Linked Data". In: *Conference on Hypertext and Social Media*. HT '12. Milwaukee, Wisconsin, USA: ACM, 2012, pp. 43–52. ISBN: 9781450313353. DOI: 10.1145/2309996.2310005. URL: https://doi.org/10.1145/2309996.2310005.

[32] Olaf Hartig, Katja Hose, and Juan Sequeda. "Linked data management". In: *Encyclopedia of Big Data Technologies*. Springer, 2018, pp. 1–7.

[33] Olaf Hartig and M. Tamer Özsu. *Walking without a Map: Optimizing Response Times of Traversal-Based Linked Data Queries (Extended Version)*. 2016. arXiv: 1607.01046 [cs.DB].

[34] Olaf Hartig and Jorge Pérez. "LDQL: A query language for the Web of Linked Data". In: *Journal of Web Semantics* 41 (2016), pp. 9–29. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2016.10.001. URL: https://www.sciencedirect.com/science/article/pii/S1570826816300476.

[35] Tom Heath and Christian Bizer. "Linked Data: Evolving the Web into a Global Data Space". In: Synthesis Lectures on the Semantic Web: Theory and Technology (2011). Ed. by James Hendler and Frank van Harmelen. URL: http://linkeddatabook.com/editions/1.0/.

[36] Katja Hose and Ralf Schenkel. "Towards benefit-based RDF source selection for SPARQL queries". In: *Proceedings of the 4th International Workshop on Semantic Web Information Management*. 2012, pp. 1–8.

[37] Farah Karim, Maria-Esther Vidal, and Sören Auer. "Compacting frequent star patterns in RDF graphs". In: *Journal of Intelligent Information Systems* 55.3 (Apr. 2020), pp. 561–585. ISSN: 1573-7675. DOI: 10.1007/s10844-020-00595-9. URL: http://dx.doi.org/10.1007/s10844-020-00595-9.

[38] Jose Emilio Labra-Gayo et al. "RDF Data integration using Shape Expressions". In: (July 2023). DOI: 10.37044/osf.io/md73k. URL: http://dx.doi.org/10.37044/osf.io/md73k.

[39] Jose-Emilio Labra-Gayo et al. *Validating and describing linked data portals using shapes*. 2017. arXiv: 1701.08924 [cs.DB]. URL: https://arxiv.org/abs/1701.08924.

[40] Peter Mechant et al. "Saving the web by decentralizing data networks? A socio-technical reflection on the promise of decentralization and personal data stores". In: *2021 14th CMI International Conference*. 2021, pp. 1–6. DOI: 10.1109/CMI53512.2021.9663788.

[41] Marios Meimaris et al. "Extended Characteristic Sets: Graph Indexing for SPARQL Query Optimization". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)* (2017), pp. 497–508. URL: https://api.semanticscholar.org/CorpusID:3535607.

[42] Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. "The Odyssey Approach for Optimizing Federated SPARQL Queries". In: *The Semantic Web – ISWC 2017*. Ed. by Claudia d'Amato et al. Cham: Springer International Publishing, 2017, pp. 471–489. ISBN: 978-3-319-68288-4.

[43] Thomas Neumann and Guido Moerkotte. "Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins". In: *2011 IEEE 27th International Conference on Data Engineering* (2011), pp. 984–994. URL: https://api.semanticscholar.org/CorpusID:2208604.

[44] Jakob Nielsen. "5.5 Feeback". In: *Usability Engineering*. Morgan Kaufmann, 1993, p. 135.

[45] Jakob Nielsen. "Response times: the three important limits". In: *Usability Engineering* (1993).

[46] Jakob Nielsen. *The Need for Speed in AI — uxtigers.com*. https://www.uxtigers.com/post/ai-response-time. [Accessed 14-07-2025].

[47] Reinhard Pichler and Sebastian Skritek. "Containment and equivalence of well-designed SPARQL". In: *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '14. Snowbird, Utah, USA: Association for Computing Machinery, 2014, pp. 39–50. ISBN: 9781450323758. DOI: 10.1145/2594538.2594542. URL: https://doi.org/10.1145/2594538.2594542.

[48] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. *Optimizing SPARQL Queries using Shape Statistics*. en. 2021. DOI: 10.5441/002/EDBT.2021.59. URL: https://openproceedings.org/2021/conf/edbt/p202.pdf.

[49] Markus Lanthaler Richard Cyganiak David Wood. *RDF 1.1 Concepts and Abstract Syntax — w3.org*. https://www.w3.org/TR/rdf11-concepts/. [Accessed 03-03-2025].

[50] R Roller, J Roes, and E Verbree. "Benefits of linked data for interoperability during crisis management". In: (2015).

[51] Alexandra Rowland et al. "Interoperability and integration: an updated approach to linked data publication at the Dutch Land Registry". In: *ISPRS international journal of geo-information* 11.1 (2022), p. 51.

[52] A. Sambra et al. "Solid : A Platform for Decentralized Social Applications Based on Linked Data". In: 2016. URL: https://www.semanticscholar.org/paper/Solid-%3A-A-Platform-for-Decentralized-Social-Based-Sambra-Mansour/5ac93548fd0628f7ff8ff65b5878d04c79c513c4 (visited on 12/18/2024).

[53] S. Sevignani. "The commodification of privacy on the Internet". In: *Science and Public Policy* 40.6 (Nov. 2013), pp. 733–739. DOI: 10.1093/scipol/sct082. URL: https://doi.org/10.1093/scipol/sct082.

[54] *Shape Trees Specification — shapetrees.org*. https://shapetrees.org/TR/specification/. [Accessed 06-12-2024]. Eric Prud'hommeaux, Justin Bingham.

[55] Mirko Spasić and Milena Vujošević Janičić. "Solving the SPARQL query containment problem with SpeCS". In: *Journal of Web Semantics* 76 (Apr. 2023), p. 100770. DOI: 10.1016/j.websem.2022.100770. URL: https://doi.org/10.1016/j.websem.2022.100770.

[56] Ashok Malhotra Steve Speicher John Arwe. *Linked Data Platform 1.0*. https://www.w3.org/TR/ldp/. [Accessed 26-02-2025]. 2015.

[57] Heiner Stuckenschmidt et al. "Index structures and algorithms for querying distributed RDF repositories". In: *Proceedings of the 13th International Conference on World Wide Web*. WWW '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 631–639. ISBN: 158113844X. DOI: 10.1145/988672.988758. URL: https://doi.org/10.1145/988672.988758.

[58] Ruben Taelman. "Towards Applications on the Decentralized Web using Hypermedia-driven Query Engines". In: *ACM SIGWEB Newsletter* 2024.Summer (2024), pp. 1–9.

[59] Ruben Taelman and Ruben Verborgh. *Link Traversal Query Processing Over Decentralized Environments with Structural Assumptions*. Ed. by Terry R. Payne et al. Cham, 2023.

[60] Ruben Taelman et al. "Components.js: Semantic Dependency Injection". In: *Semantic Web Journal* (Jan. 2022). URL: https://linkedsoftwaredependencies.github.io/Article-System-Components/.

[61] Ruben Taelman et al. "Comunica: a Modular SPARQL Query Engine for the Web". In: *Proceedings of the 17th International Semantic Web Conference*. Oct. 2018. URL: https://comunica.github.io/Article-ISWC2018-Resource/.

[62] Bryan-Elliott Tam et al. "Opportunities for Shape-based Optimization of Link Traversal Queries". In: *Proceedings of the 16th Alberto Mendelzon International Workshop on Foundations of Data Management*. Sept. 2024. URL: https://arxiv.org/pdf/2407.00998v2.

[63] Bryan-Elliott Tam et al. "Optimizing Traversal Queries of Sensor Data Using a Rule-Based Reachability Approach". In: *Proceedings of the 23rd International Semantic Web Conference: Posters and Demos*. Nov. 2024. URL: https://arxiv.org/pdf/2408.17157.

[64] Tiziana Terranova. "Free Labor: Producing Culture for the Digital Economy". In: *Social Text* 18 (2000), pp. 33–58.

[65] Ruben Verborgh et al. "Triple pattern fragments: a low-cost knowledge graph interface for the web". In: *Journal of Web Semantics* 37 (2016), pp. 184–206.

[66] W3C. *SPARQL Queries to Validate Shape Expressions (informative)*. 2013. URL: https://www.w3.org/2013/ShEx/toSPARQL.html (visited on 09/11/2023).

[67] W3C. *SPARQL Query Language for RDF - Formal Definitions — w3.org*. https://www.w3.org/2001/sw/DataAccess/rq23/sparql-defns.html. [Accessed 04-03-2025]. 2001.

[68] Mark D. Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3.1 (Mar. 2016). ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18. URL: http://dx.doi.org/10.1038/sdata.2016.18.

[69] Jeff Zucker et al. *Type Indexes — solid.github.io*. https://solid.github.io/type-indexes/. [Accessed 07-04-2025]. 2023.

Appendix

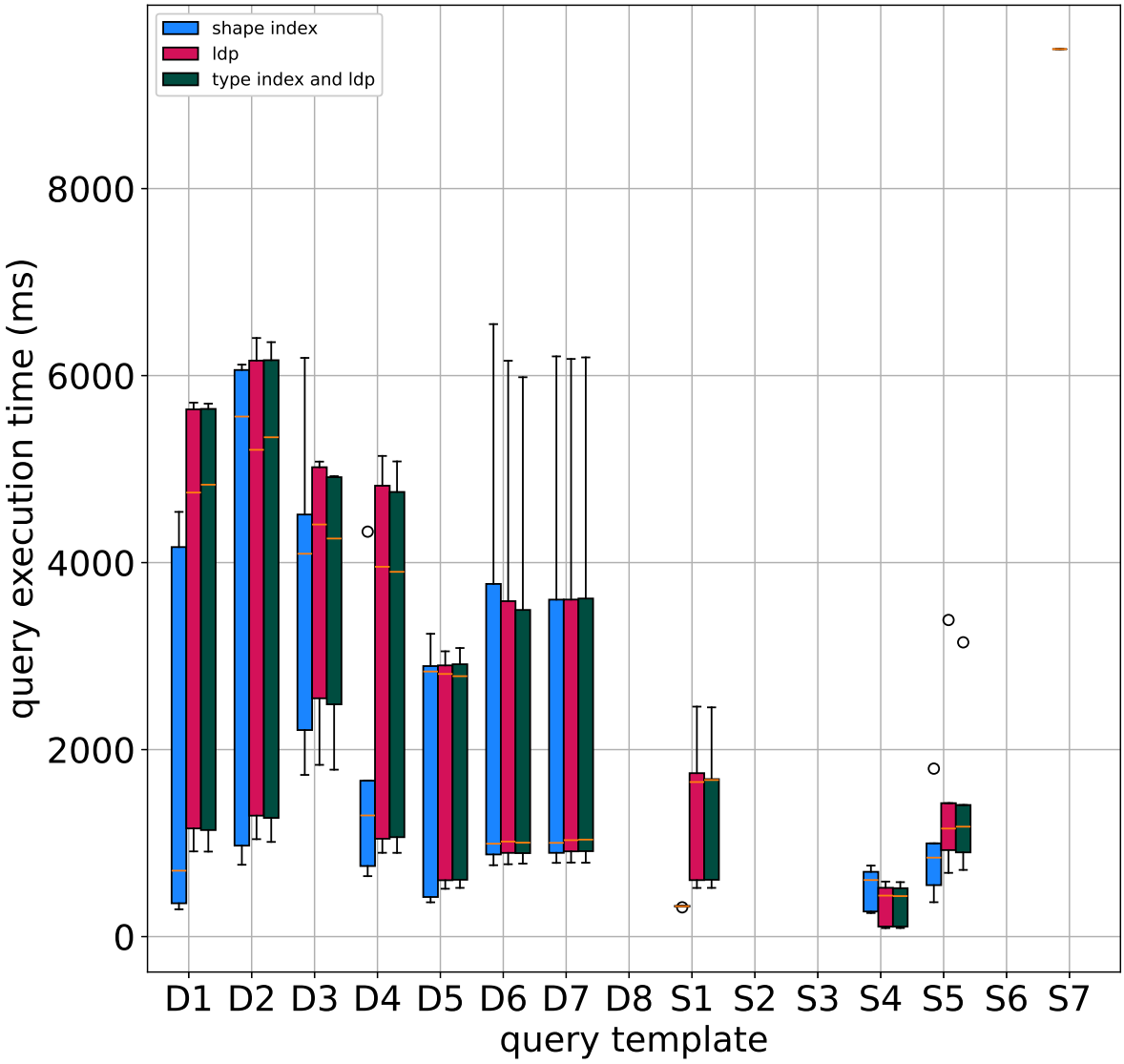

Figure 12. The query execution time of the different approaches grouped by query template.

query template	relation execution time	p-value	average ratio HTTP request
D1	lesser	1.14E-36	0.57
D2	lesser	4.42E-04	0.88
D3	similar	7.47E-01 (RH)	0.97
D4	lesser	2.07E-17	0.65
D5	lesser	5.58E-03	0.88
D6	similar	2.56E-01 (RH)	1.12
D7	similar	7.83E-01 (RH)	1.12
S1	lesser	1.12E-83	0.33
S4	greater	3.76E-22	13.00
S5	lesser	3.12E-17	0.44

Table 2. Table comparing the shape index approach to the state-of-the-art. RH, indicate that the p-value is associated to the rejected hypothesis. Every query performs better or similarly to the state-of-the-art with the shape index approach except for interactive-short-4. One might expect the average ratio of HTTP requests for D6 and D7 to be one. In our implementation, however, the query subsumption algorithm naively retrieved nested shapes, leading to some shapes being dereferenced multiple times. This is an implementation artifact rather than an inherent property of the method. Even in this worst case, the impact on the number of HTTP requests is small and does not affect the overall conclusions, especially given that requests are sent in parallel.

Template	**Shape Index**			**Type Index**			**LDP**		
	IT ($n \times ms$)	**SFT** ($n \times ms$)	**LRT** ($n \times ms$)	**IT** ($n \times ms$)	**SFT** ($n \times ms$)	**LRT** ($n \times ms$)	**IT** ($n \times ms$)	**SFT** ($n \times ms$)	**LRT** ($n \times ms$)
D1	0.00	**1.70**	**16.96**	0.00	0.22	11.22	0.00	0.21	10.67
D2	0.00	**1.91**	119.54	0.00	0.31	119.75	0.00	0.23	**124.94**
D3	0.00	0.00	85.14	0.00	0.00	**97.01**	0.00	0.00	85.78
D4	0.00	**0.36**	**10.25**	0.00	0.06	1.92	0.00	0.07	1.75
D5	0.00	**1.03**	**13.82**	0.00	0.74	12.18	0.00	0.74	12.16
D6	0.00	**0.08**	16.08	0.00	0.08	**20.04**	0.00	0.08	18.88
D7	0.00	**0.08**	**0.28**	0.00	0.07	0.20	0.00	0.07	0.21
S1	0.00	**0.68**	**1.25**	0.00	0.18	0.28	0.00	0.18	0.27
S4	0.00	0.49	0.00	**0.01**	**0.66**	**0.04**	0.01	0.66	0.04
S5	0.00	0.25	0.05	0.00	0.34	0.10	0.00	**0.37**	**0.13**
S7	**0.00**	**0.00**	**0.00**	-	-	-	-	-	-

Table 3. Table showing the average diefficiency metric for different approaches across query templates, measured at three time thresholds: instantaneous time (IT, 0.1s), seamless flow time (SFT, 1s), and at the time of the last result (LRT). The diefficiency is expressed as the number of results (n) per millisecond (ms). For most queries, diefficiency at IT is 0, as no results are produced quickly enough. At SFT, the shape index approach generally yields better performance, while at LRT, results tend to converge across approaches.

Template	**Shape Index**			**Type Index**			**LDP**		
	FT (s)	**TT** (s)	**WT** (s)	**FT** (s)	**TT** (s)	**WT** (s)	**FT** (s)	**TT** (s)	**WT** (s)
D1	**0.66**	**0.11**	0.23	1.51	1.09	**0.21**	1.55	1.02	0.21
D2	**1.25**	0.00	0.78	1.91	**0.00**	0.39	1.88	0.00	**0.38**
D3	3.74	**0.00**	0.00	**3.67**	0.00	0.00	3.77	0.00	0.00
D4	**1.74**	0.00	0.00	3.14	0.00	0.00	3.17	**0.00**	0.00
D5	**0.49**	**0.03**	0.22	1.17	0.04	0.01	1.18	0.04	**0.00**
D6	1.42	**0.02**	0.58	**1.40**	0.04	**0.49**	1.45	0.07	0.49
D7	**1.48**	1.19	0.00	1.56	1.12	0.00	1.55	**1.11**	0.00
S1	**0.32**	**0.00**	0.00	1.30	0.09	0.00	1.31	0.09	0.00
S4	0.51	**0.01**	0.00	**0.34**	0.01	0.00	0.34	0.01	0.00
S5	0.91	**0.00**	0.00	0.78	0.68	0.00	**0.75**	0.76	0.00
S7	-	-	**0.00**	-	-	-	-	-	-

Table 4. Continuous performance metrics showing First Result Arrival Time (FT), Termination Time (TT), and Waiting Time (WT). FT refers to the time taken for the first result to be delivered. TT captures the delay between the last result and the termination of the query engine. WT represents the total time the user waited more than one second between consecutive results, indicating interruptions in the seamless flow experience.

Query Template	avg (ms)	std (ms)	max (ms)
D1	0.062	0.073	0.539
D2	0.106	0.461	4.655
D3	0.040	0.034	0.133
D4	0.043	0.068	0.672
D5	0.015	0.015	0.025
D6	0.068	0.039	0.174
D7	0.041	0.101	1.005
D8	0.033	0.045	0.426
S1	0.074	0.064	0.560
S2	0.309	0.327	2.791
S3	0.072	0.069	0.650
S4	0.042	0.032	0.144
S5	0.043	0.057	0.551
S6	0.132	0.343	3.473
S7	0.176	0.158	0.971

Table 5. Query-Shape containment computation time (100 samples) is negligeable with the most restrictive shapes of our experiments.