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

Journal Title
XX(X):1–25
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Bryan-Elliott Tam[1], Joachim Van Herwegen[1], Pieter Colpaert[1], Ruben Verborgh[1] and Ruben Taelman[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 by keeping data closer to its source, yet efficient query processing remains a significant challenge. Link Traversal Query Processing (LTQP) enables querying across decentralized networks, however, it often suffers from long execution times and high data transfer costs due to the large number of Hypertext Transfer Protocol (HTTP) requests involved. In many scenarios, queries are highly selective with respect to the data model objects distributed across the network. For example, in a social media application where users store heterogeneous data, a query may focus solely on the posts and comments created by users, without requiring any of their additional user information. We refer to such queries as *data-model selective*. 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 for such queries. 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 substantially improves query execution time, first-result arrival time, diefficiency, and network usage for data-model selective queries, while having unsignificative impact on non-selective data-model queries. These gains come at the cost of only a minor increase in the number of triples per shape-index instance. Moreover, our approach is resilient, retaining performance benefits even in networks where some data providers do not supply shape-index information. This work demonstrates that shape-based metadata can significantly optimize LTQP in decentralized knowledge graphs for an important class of queries. By exposing such metadata, data providers not only enhance data quality and interoperability but also improve the efficiency of traversal-based query processing.

Keywords

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

Introduction

Multiple studies have highlighted that social web applications relying on highly centralized data management systems give rise to problems of ownership, threats to democracy, reinforcement of inequality, and antagonism between users and owners [67, 16, 56, 43]. Decentralizing data over the web is a response to these problems and an integral component of initiatives focused on data sovereignty, even if several authors consider it an insufficient solution on its own [43, 16]. Linked Data and knowledge graphs (KGs) [38] are technical contributions toward such a decentralized web of data. However, SPARQL, the standard query language for Resource Description Framework (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) [34] is a query paradigm designed for querying non-indexed, Decentralized Knowledge Graphs (DKGs) on the web, by leveraging the descriptive power of Internationalized Resource Identifier (IRI)[2] 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 Hypertext Transfer Protocol (HTTP) requests as demonstrated by Hartig and Özsu [36]. From another perspective, it has been shown that in Decentralized Environments with Structural Properties (DESPs), it is possible to attain query completeness for various types of practical queries with acceptable execution times for the context of social media applications [62, 48]. Such environments are decentralized networks in which data providers expose implicit or explicit data structures, referred to as *structural properties*, that query engines can leverage during traversal. Structural properties ensure data discoverability, which in turn helps guarantee result completeness.

[1]Department of Electronics and Information Systems, Ghent University – imec

Corresponding author:
Bryan-Elliott Tam
Email: bryanelliott.tam@ugent.be

[2]An IRI [21] is a unique sequence of characters that identifies a resource on the web; it generalizes the Uniform Resource Identifier (URI) [7] by additionally permitting characters from the Unicode set.

In practice, DESPs emerge in various contexts, such as social networks [62] and the publication of sensor data [66], among others. The work on the topic of DESPs for LTQP [62, 10] suggests that various optimizations are feasible for LTQP in such environments, in contrast to Hartig and Özsu [36], who concluded that the large search space of LTQP offers limited opportunities for optimization.

In general, data on the World Wide Web lack 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. Thus, for the vast majority of queries, traversal query engines dereferences a large proportion of resources that are not query-relevant, as depicted in Figure 1. 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 [10]. In this work, we extend a dataset summarization approach for decentralized environments known as the *shape index* [65]. 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. Whereas existing traversal methods focus on discovering which links to follow, our method prunes links that can be proven irrelevant, removing even a discovered link when the data shapes show it cannot contribute to the query results. 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 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.** An example of such a query is one that requests messages within a user's subweb over a specified date range, while also incorporating contextual information from outside the subweb. We describe this category of network-tied queries as *data-model selective*. In this context, *data-model objects* are templates of concepts represented in RDF, such as messages, posts, or proteins. Accordingly, we consider RDF data shapes to be formal descriptions of these data-model objects. For the purposes of this paper, we define a *data-model selective query* as a query whose evaluation requires fewer than 20% of the data-model objects within the network. In this work, we do not aim to optimize queries that retrieve a large proportion of a subweb's data model objects. For instance, queries that request all user information published after a given date are outside the scope of our optimization; however, we attempt to ensure that the performance of such queries is not degraded.

Our contributions are threefold:

Method. A novel shape-based pruning method for LTQP that introduces the following new notions:

(i) *result-based completeness*, formalized as a search-space minimization problem;

(ii) *composite reachability criteria* that combine link discovery and pruning;

(iii) a network abstraction of *subwebs* and DKGs; and

(iv) *shape-based link pruning*, which uses RDF data shapes to discard sources that cannot contribute to the query results while preserving result completeness.

Software components. Three artifacts[3] realize the method:

(i) a web specification for the *shape index* data summary;

(ii) a *query-shape subsumption algorithm* for assessing data source relevance;

(iii) an open-source implementation in the Comunica SPARQL query engine [64].

Evaluation. An extensive evaluation on the SolidBench benchmark, across multiple metrics (query execution time, first-result arrival time, diefficiency, and number of HTTP requests), comprising:

(i) a comparison against state-of-the-art LTQP traversal (the type index and Linked Data Platform (LDP) approaches), showing that shape-based pruning reduces the number of HTTP requests and improves query execution time for data-model-selective queries, and even enables a query that no prior approach could complete, while leaving non-selective queries essentially unaffected;

(ii) a cost analysis of the query-shape subsumption algorithm, showing that its overhead is negligible relative to query execution;

(iii) a resilience study under partial and reduced shape-index coverage, showing that the performance gains persist even when not all data providers expose a shape index.

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 decompose it along the four axes that structure our evaluation:

Effectiveness. Whether pruning improves performance for the targeted queries without harming the others:

H1 The shape indexes approach reduce the number of non-contributing data sources retrieved and the query execution time of *data-model selective* queries, while preserving result completeness.

H2 The shape indexes approach does not affect the query execution time of *non-data-model selective* queries.

[3] The shape index specification and the open-source implementations of all software components are available online: https://constraintautomaton.github.io/shape-index-specification/ and https://github.com/shapeIndexComunicaExperiment/documentation.

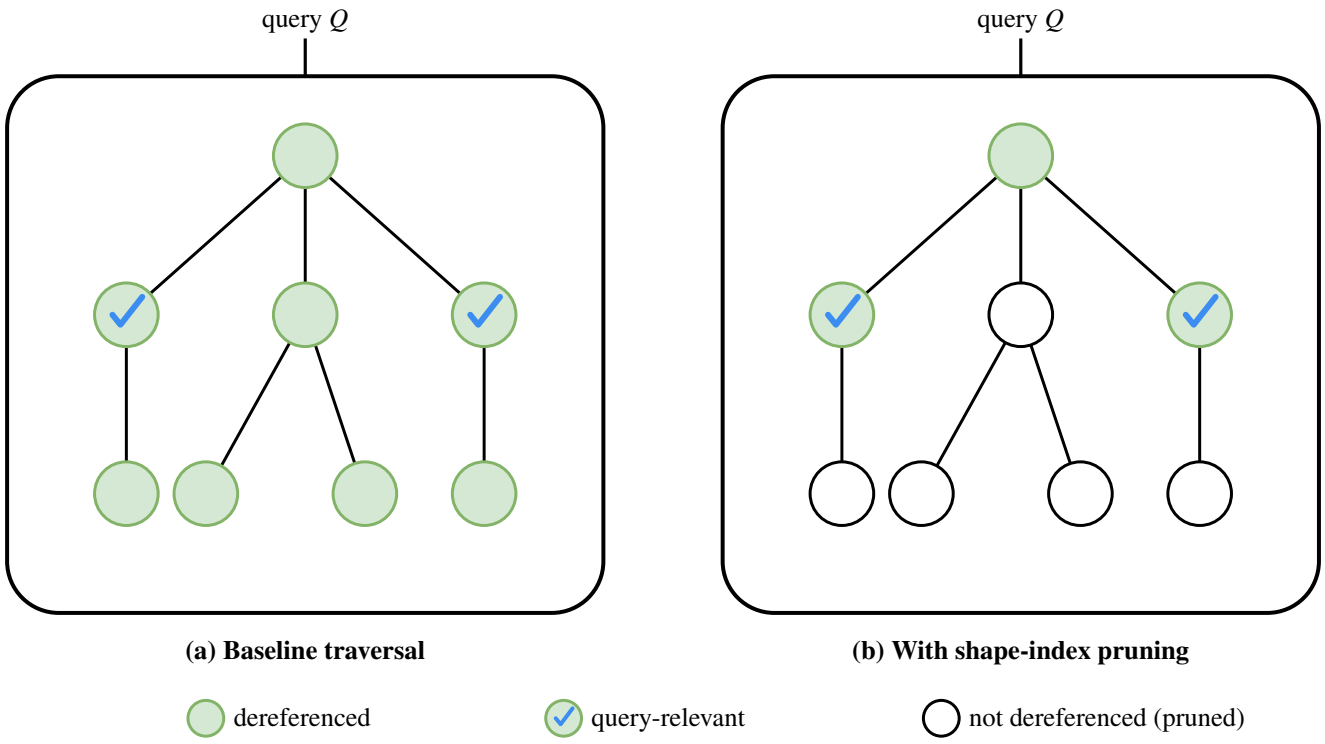

Figure 1. The same query Q is issued to a network under two strategies. (a) A baseline link-traversal engine dereferences many sources (green) even though only a few are query-relevant (blue check), wasting requests on irrelevant ones. (b) Our shape-index pruning dereferences only the entry point and the query-relevant sources, leaving the sources that are provably irrelevant not dereferenced (white). The goal of our work is to minimize the number of query-irrelevant sources that are dereferenced by a link traversal query engine.

Cost. Whether the pruning mechanism is itself inexpensive.

H3 The execution time of a query-shape subsumption algorithm is negligible compared to the query execution.

Robustness. How the amount of shape-index information across the network affects the performance:

H4 Stricter shape constraints lead to a greater reduction in HTTP requests.

H5 Querying a network with more *complete* shape indexes results in faster query execution.

H6 Performance gains can be achieved even in networks with partial shape index coverage, demonstrating that the method provides significant improvements without requiring all network participants to adopt shape indexes.

Explainability. What relationship explains the observed gains:

H7 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 the paper.

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, whereby the engine dereferences the IRIs found in retrieved documents to discover further documents to query [36]. 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. A federation is a set of queryable RDF sources, typically SPARQL endpoints, that is fixed and known prior to query execution. 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, 33], 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) [68] focus on the efficient partitioning of a known dataset, while LTQP focuses on the efficient discovery of an unknown dataset. More generally, an LDF interface publishes a dataset as a set of queryable fragments over the Web, balancing query-processing effort between the server and the client [69].

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 [34]. A traversal is complete when it dereferences every link in this set, so that the query is answered completely over the subgraph those links induce; completeness in LTQP is therefore relative to the set of reachable links rather than to the entire Web, whose search space is too large to traverse exhaustively. The first method used to define this set was the *reachability criteria* [34], boolean functions that determine whether a given link should be dereferenced. In practice, the query engine iterates over all triples in its internal data source and applies the reachability criteria to each IRI appearing in those triples. These criteria are defined internally by the engine, giving designers considerable freedom in how they are implemented and which links are dereferenced. However, there is currently no standard mechanism for users to specify them, nor for query engine developers to adopt a common approach. Building on this, the theoretical query language Linked Data Query Language (LDQL) [37] was introduced, which separates the traversal definition from the query definition. It provides a standard mechanism for query engines to define their traversal logic, while opening the possibility for users to provide a traversal policy alongside their queries. Further advancements include the Subweb Specifications Language (SWSL) [10], which allows data providers to define how their DKG should be traversed. Inspired by SWSL, traversal-based querying has utilized the LDP and the Type Index specification [62]. LDP traversal follows all links within a data space, while Type Index traversal uses mappings from RDF types to relevant resources [73], allowing queries to prioritize implicitly relevant data sources. Link prioritization has also been studied, however, it has been shown to not improve query execution time [36], even in DESPs [23]. 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. We distinguish discovery from pruning as assertion differs from negation: a discovery criterion expresses that a link should be explored, whereas a pruning criterion expresses that it should not, and the latter takes precedence, so a pruned link is not followed even when a discovery criterion would select it. Discovery builds the search domain of an LTQP query, whereas pruning carves holes into that domain. Although both can be described as retaining a subset of links, only inclusion is sound under the open-world assumption of the web, since a missing link is no evidence that relevant data is absent; excluding links soundly requires closed-world knowledge about a region of the web. 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 Shapes

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 [26, 29]. The two most well known RDF data shape formalisms are Shapes Constraint Language (SHACL) [28] and Shape Expressions (ShEx) [29]. For common use cases, they are equally expressive and interchangeable [27]. RDF data shapes have already been used in the literature for querying centralized KGs [51, 18]. Shape Trees [57] are an index structure for validating and organizing DKGs; however, to the best of our knowledge, they have not been used for query optimization, and their *virtual hierarchy* [57] makes it challenging for a query engine to efficiently capture the relationship between a resource IRI and its corresponding shape. Beyond this conceptual mismatch, Shape Trees were not yet practical to build upon at the time of this study: the specification is not widely adopted, and its supporting documentation and tooling were not sufficiently mature or stable. We therefore conduct our investigation using the Shape Index specification [65] to map shapes to KGs. Importantly, our findings are not tied to this particular summary: our approach relies in essence on a mapping between shapes and sets of IRIs, so any structure providing such a mapping—including a more mature Shape Trees—could serve as the underlying summary instead. Additionally, automatic generation of RDF data shapes from KGs [25] and shape-based data integration [41] have been studied and can support shape-based summary approaches for DKGs.

Source Selection

Source selection is a crucial challenge in decentralized querying [39, 32]. It is the process of determining, before or during query execution, which of the available data sources can contribute to the result of a query, so that sources that cannot are not accessed. 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, Vocabulary of Interlinked Datasets (VoID) descriptions, and histogram techniques have been explored in the context of federated querying [39, 32, 45]. A VoID description is an RDF document that summarizes a dataset, reporting, among other things, its total number of triples and the set of predicates it uses. Figure 2 shows such a description for the `sw:posts` dataset of our running example.

However, most of those source selection methods face the limitation of assuming a small number of data sources [32], leaving their suitability for LTQP uncertain. Bloom filters [19] are also a mechanism that has shown success for federated DKGs, yet in the context of LTQP, it has been shown that bloom filters have little effect on performance [31]. Schema-based indexing using ontologies [60] 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 [32], 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 [46, 44, 45]. However, their applicability to LTQP has not been

```
@prefix void: <http://rdfs.org/ns/void#> .
@prefix ex:   <http://example.com/> .
@prefix sw:   <http://subweb/> .

sw:posts a void:Dataset ;
    void:entities 160 ;
    void:triples 480 ;
    void:propertyPartition [
        void:property ex:creator ;
        void:triples 160 ] ;
    void:propertyPartition [
        void:property ex:content ;
        void:triples 160 ] ;
    void:propertyPartition [
        void:property ex:reply ;
        void:triples 160 ] .
```

Figure 2. A short VoID description of the `sw:posts` dataset from the running example of Figure 4, describing 160 posts that together amount to 480 triples, partitioned over the `ex:creator`, `ex:content`, and `ex:reply` predicates.

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 KGs using the SPARQL query language [71]. 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 triples $t = (s, p, o)$ 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, except that any of the three positions may additionally be a variable from $\mathcal{V}$, the set of every possible variable, where $s_{tp} \in \mathcal{IBV}$, $p_{tp} \in \mathcal{IV}$ and an object term $o_{tp} \in \mathcal{IBVL}$.*

We also define two access functions to respectively get the subject and object term of a triple or a triple pattern while ignoring literals, $S : (\mathcal{IBV}, \mathcal{IV}, \mathcal{IBVL}) \to \mathcal{IBV}$ and $O : (\mathcal{IBV}, \mathcal{IV}, \mathcal{IBVL}) \to \mathcal{IBV}$, such that for a triple or triple pattern (s, p, o), $S((s, p, o)) = s$ and $O((s, p, o)) = o$ when $o \in \mathcal{IBV}$, while O returns nothing when o is a literal. 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 [34]. To formalize the completeness of queries, *Reachability criteria* [34] 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 three parameters: an RDF triple t from an internal triple store, a dereferenceable IRI iri occurring in t, and a union of conjunctive queries Q. Their domain and range are $c_i : \big((\mathcal{IB}, \mathcal{I}, \mathcal{IBL}),\ \mathcal{I},\ \mathcal{Q}\big) \to \{\text{true}, \text{false}\}$, where $\mathcal{Q}$ is the set of unions of conjunctive queries. If c_i returns $true$, the query engine must dereference iri. Dereferencing an IRI means issuing an HTTP request to it to retrieve a resource. More formally

$$c_i(t, iri, Q) \to \{\text{true}, \text{false}\} \tag{1}$$

Decentralized Knowledge Graphs and Subweb

We define a DKG as a single KG G materialized in a network of resources R. A resource $r_i \in R$ is a document that materializes exactly one KG $g_i \subseteq G$, which is a set of triples [52]. We denote this mapping $r_i \mapsto_{\mathcal{G}} g_i$. A resource is exposed by an IRI iri_i, an identifier pointing to it, denoted by $iri_i \mapsto_{\mathcal{R}} r_i$; the resource is thus distinct from the IRI that identifies it. The network forms a graph where the resources r_i are the nodes and the iri_j are directed edges starting from r_i to r_j. The iri_j are RDF terms in the triples in g_i. G is formed by the union of all the g_i mapped to a resource in the network. A subweb is a (sub)DKG defined by the KG derived from a set of IRIs controlled by a data provider.

Data-model Selectiveness

Data-model selectiveness is an ordering of queries over networks. A *data-model object* is a well-defined template of sub-KGs that corresponds to a single data type, such as a user or a post. By *well-defined*, we mean that there exist explicit rules which deterministically specify the constraints on the triples within the data-model object, as well as its boundary within a larger KG. In this work, we assume that data-model objects are described using RDF data shapes, however, they could also be described using vocabularies or other formalisms. We define DM as the set of the data-model objects present in the networks. We define a function to determine all the relevant data-model objects for a given query:

$$P(Q) = \{D \mid (D \in DM) \textbf{ is query relevant for } Q\}. \tag{2}$$

A data-model object D is query relevant for Q when at least one conjunct of Q can be answered by a KG derived from D. A query Q_i is more data-model selective than another query Q_j if and only if

$$|P(Q_i)| < |P(Q_j)|\,. \tag{3}$$

For practical use cases, we may consider only the data-model objects present in a single subweb s rather than the whole DKG, replacing DM by DM^s, the set of data-model objects in subweb s, and P by the corresponding function $P^s(Q)$. Furthermore, for practical designation, we say that a query is *data-model selective* (in absolute terms) if

$$\frac{|P^s(Q)|}{|DM^s|} \leq 0.2, \tag{4}$$

that is, if at most 20% of the data-model objects are potentially relevant to the query.[4]

[4] The 0.2 threshold is arbitrary. There is no objective criterion for choosing such a value, so we picked it solely for this practical designation.

Approach

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

To illustrate this approach, we present the example in Figure 3, 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. Figure 4 illustrates an example of a shape index alongside a query pattern that our approach analyzes; both are formalized in the following subsections.

Result-Based Completeness in LTQP

Our approach of pruning in LTQP focuses 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 KG $G'' \subseteq G'$, 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

$$[\![Q]\!]^{G''} = [\![Q]\!]^{G'} \tag{5}$$

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 [36], can dominate execution time. Therefore, reducing HTTP requests provides a twofold benefit: fewer resources to process and faster query execution.

Example In Figure 3, the feature query starts at the seed document of Subweb 3. The discovery criterion C' follows every edge reachable from that document and therefore traverses the whole network (17 links). The more selective C'' follows only the edges leading to posts and their replies, traversing a subset of the network (4 links). The two criteria build different graphs G' and G'', yet report the same answer to Q (Equation 5 holds), needing only 4 dereferences instead of 17.

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:

$$SI = \{s_1 \mapsto IRI_1 s_2 \mapsto IRI_2 \cdots, s_n \mapsto IRI_n\} \tag{6}$$

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 $D_{SI} = \bigcup_{IRI \in \text{codomain}(SI)} IRI$. We denote a shape index as *complete* when every shape $s_i \in \text{dom}(SI)$ has a closed world assumption [28, 29] [5] 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 [6] and an example of serialization in Figure 4.

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 associated with a shape index follow the same RDF data shape. We call these entities graph stars, an extension of the RDF star patterns concept [40]. Defined in Definition 4, 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 such that the objects of triples in one star pattern act as subjects in others, 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 targets 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 4 illustrates an example of a graph star pattern.

Definition 3. Star Pattern. *Given a query Q, we define a star pattern Q_{star} as a set of $tp \in Q$ [40] with the same*

[5] We also refer to those shapes as closed.

[6] https://constraintautomaton.github.io/shape-index-specification/

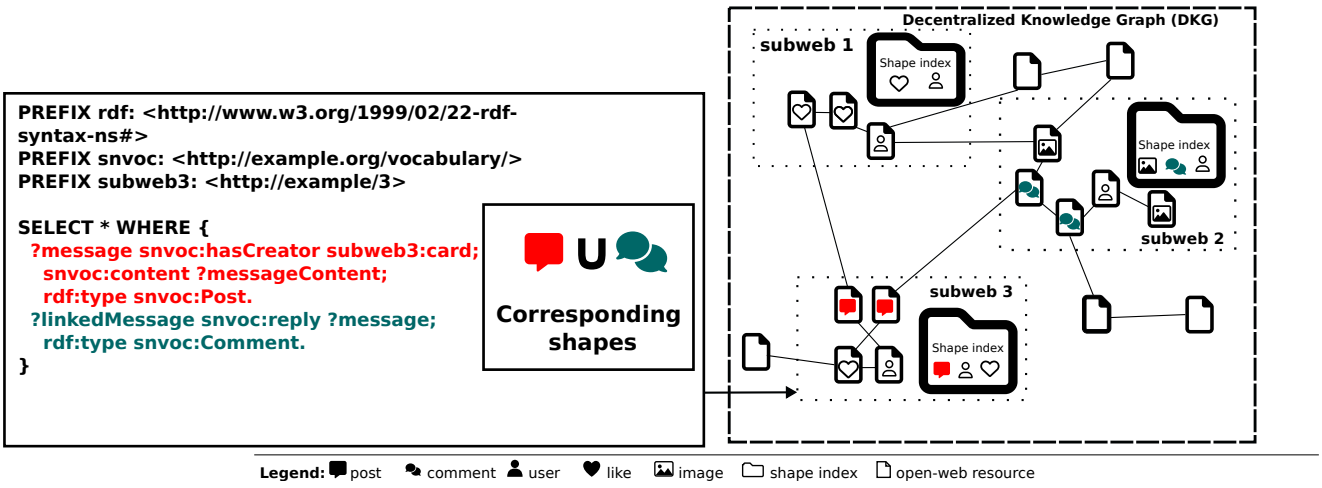

Figure 3. When resources of a DKG are indexed with a shape index, a query engine can dereference a subset of the network. Each node is an RDF resource, a document identified by an IRI, and each edge is an IRI in one resource that links to another. The icon embedded in a node indicates the shape, and thus the data type such as a post, comment, or user, that the resource conforms to; the legend at the bottom of the figure defines these icons. Each subweb publishes a shape index, drawn as a folder, that maps the shapes its resources conform to, so Subweb 1 contains user and like resources, Subweb 2 user, comment, and image resources, and Subweb 3 post, user, and like resources. Documents outside the subwebs are resources from the open web and are thus not bound by any shape. The query engine starts its query at Subweb 3, and the colored documents are those relevant to the query shown on the left: red for posts and green for comments.

```
@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 4. On the left, an example illustrates a shape index that maps a set of IRIs, represented using a URI template [30], 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.

subject such that given a builder function

$$BQ_{star}(s) = \{tp_i \in Q \mid S(tp_i) = s\} \tag{7}$$

with $s \in \mathcal{IBV}$ *then* $Q_{star} = BQ_{star}(s)$.

Definition 4. Graph Star Pattern (GSP) and Graph Star. *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* $O_{star} : q \in Q \rightarrow (\mathcal{IBV})^2$ *that returns every non-literal object terms of a star pattern.*

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

$$Q_{starG} = \bigcup_{q \in Q_{starGT}} q \tag{8}$$

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

$$q_i = \begin{cases} \{Q_{star_s}\} & \text{if } i = 1 \\ \left\{ BQ_{star}(o) \mid o \in \bigcup_{q \in Q_{starGT_{i-1}}} O_{star}(q) \right\} & \text{if } i > 1 \end{cases} \tag{9}$$

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

Practical Considerations for Shape Indexes 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 [11], as they do not include detailed statistics such as triple counts. Nonetheless, VoID descriptions have been successfully employed for query optimization in federated queries [45].

Exposed schemas can also enhance interoperability, which is important in many application domains [54, 9, 8, 53]. 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 [25] can facilitate their creation. Entries in shape indexes correspond to sets of IRIs, which can be structured using URI templates [30], reducing the need for exhaustive redefinition. For prescriptive shapes, contributions in shape-based data integration [41] 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 Findable, Accessible, Interoperable, and Reusable (FAIR) data principles [72] 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 10 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.

$$C(t, iri, Q) = \bigvee_{cd \in Cd} cd(t, iri, Q) \wedge \bigwedge_{cp \in Cp} cp(t, iri, Q) \tag{10}$$

This structure reflects the distinction between discovery and pruning: discovery criteria are combined disjunctively, so any one of them suffices to assert that a link is reachable, whereas pruning criteria are combined conjunctively, so a single pruning criterion blocks a link regardless of how many discovery criteria assert it. Here the closed-world knowledge that licenses pruning is supplied by the closed shapes of a shape index, which let the engine soundly conclude that a region of the web cannot contribute to the query results.

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 5 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, given a query Q,

$$IRI_p = \left\{ \bigcup SI_i(s_j) | Q \not\sqsubseteq_{qs} s_j \wedge s_j \in \text{dom}(SI_i) \right\}$$

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

$$cp_{si}(t, iri, Q) = iri \notin IRI_p \tag{11}$$

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

Example In Figure 3, the engine's criterion C combines a discovery criterion $cd_{\text{shape index}}$ that follows the links to each subweb's shape index (taken disjunctively) with a pruning criterion cp_{si} (taken conjunctively). The shape index of Subweb 3 maps the post, user, and like shapes. The query subsumes the post shape but neither the user nor the like shape, so IRI_p collects the user and like resources; cp_{si} prunes exactly those, while the post resources, for which every cp holds and at least one cd holds, stay reachable. The same check prunes all of Subweb 1, whose like and user shapes are not subsumed, and in Subweb 2 every resource except the comments that reply to the posts.

Limitations 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 leveraged against the method exploiting VoID descriptions [45]. 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 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 [7] [42, 15, 70, 18]. 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 $\sqsubseteq_{qs} Q_s$, 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 is equivalent to or a specialization

[7] We only consider shapes that can be transformed into a single `SELECT` SPARQL query [15].

of those in the Q_s. Thus, the problem diverges from traditional query containment and query subsumption [58, 50] under set semantics. Query-shape subsumption does not consider only the set of solution mappings but also the constraints of the queries. [8] The complexity of the problem is reduced by the fact that Q_s has a GSP structure where predicates are always IRIs. This structure arises because shapes describe constraints on predicate and object terms of sets of triple. 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 [20, 12], making this algorithm applicable in practice.

More formally, for a S to subsume 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

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

where Q_{body} denote the Basic Graph Pattern (BGP) of the query, and $Q_{\text{unions}} = \bigcup Q_u$ represent the Union Graph Patterns (UGPs) [71] expressed in normal form, meaning each Q_u is of the form $q_i \cup q_j$, where each q is a BGP contain no union statements.

Example The query in Figure 3 has two graph star patterns. The `?message` star captures a subset of a post's predicates (`snvoc:hasCreator`, `snvoc:content`) and requires `rdf:type snvoc:Post`; to decide $Q \sqsubseteq_{qs} S$ for the post shape of Subweb 3, that shape is translated into $Q_s = T(S)$, and every triple pattern of the `?message` star is equivalent to or a specialization of a pattern in Q_s while each solution extends to one of Q_s, so the star is subsumed and the post resources are query-relevant. The `?linkedMessage` star captures a comment that must reply to a post (`snvoc:reply ?message`) and be of `rdf:type snvoc:Comment`; it is subsumed analogously by the comment shape of Subweb 2.

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 if 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. 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}. In this case, the algorithm recursively applies the same procedure to this partial GSP. To avoid cycles and redundant evaluations we maintain a set of evaluated answers in $Eval_{star}$. To solve $Q \sqsubseteq_{qs} S$, 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 procedure is detailed in Algorithm 2. 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 traverses 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:

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

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

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

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

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 GSP of the query and applying the $subsums_{\text{graphstar}}$ algorithm. Thus, the time complexity is given by:

$$O\left(n_{sunion} \cdot n^2_{tp_{max}} \cdot n_{Q_{star_{max}}} \cdot n_{Q_{starG}}\right) \quad (12)$$

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}, and $n_{Q_{starG}}$ is the number of Graph Pattern Structures (GPS) in the query Q. [9]

[8] In this sense, it bears some similarity to query containment under bag semantics, particularly through the notion of "goals-onto" containment mappings [14, 2].

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

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

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

if $M(S_{star}(Q_{star})) \in Eval_{star}$ **then**
  return $M(S_{star}(Q_{star}))$
end if
for all $tp \in Q_{star}$ **do**
  if not $match(tp, Q_{\text{sbody}})$ **then**
    $hasOnePath \leftarrow$ **false**
    for all $q_{us} \in Q_{\text{sunions}}$ **do**
      if $subsums_{\text{graphstar}}(Q_{star}, Q_{starG}, q_{us}, Eval_{star})$ **then**
        $hasOnePath \leftarrow$ **true**
      end if
    end for
    if not $hasOnePath$ **then**
      $Eval_{star} \leftarrow Eval_{star} \cup (M(S_{star}(Q_{star})) \mapsto \text{false})$
      return false
    end if
  else
    if $O(tp) \in S_{star}(Q_{starG})$ **then**
      $Eval_{star} \leftarrow Eval_{star} \cup (M(S_{star}(Q_{star})) \mapsto \text{true})$
      if not $subsums_{\text{graphstar}}(Q_{star_{O(tp)}} \in Q_{starG}, Q_{starG}, Q_s, Eval_{star})$ **then**
        $Eval_{star} \leftarrow Eval_{star} \cup (M(S_{star}(Q_{star})) \mapsto \text{false})$
        return false
      end if
    end if
  end if
end for
return true

Algorithm 2 Check if a query Q subsumes a Q_s ($subsums_{\text{Q}}$)

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

for all $Q_{starG_s} \in Q$ **do**
  for all $Q_{star_s} \in Q_{starG_s}$ **do**
    if $subsums_{\text{graphstar}}(Q_{star_s}, Q_{starG_s}, \emptyset)$ **then**
      return true
    end if
  end for
end for
return false

Experimental Evaluation

We implemented our approach using the LTQP version of the Comunica query engine [64]. We chose Comunica due to its modularity [63] and its established use in several LTQP studies [10, 62, 22, 31, 24, 65]. All implementations are open-source and are provided in the supplementary material. Similarly to other LTQP studies, we used SolidBench [62], which is based on the Linked Data Benchmark Council (LDBC) social network benchmark [4], to evaluate our contribution. Furthermore, SolidBench was developed because no LTQP benchmark existed prior to its creation [35, 62], 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, user settings, varia data (unstructured data), cities, and likes. The datasets are Solid Pods [55, 17]. In this paper, we consider a Solid Pod as a web-based file system that follows the LDP specification [59]. Each Solid Pod, alongside contains 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. Queries may range from being highly selective with small result sets to broad with larger result sets, and in some cases can also be characterized as data-model selective. 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. A description of each query template used in our evaluation is provided in Table 7 in the appendix.

To evaluate our approach, we conducted the following experiments:

- We 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 [62], and another using the LDP specification [62], in a network where each Solid Pod provides a complete shape index. [10]
- We analyze the relationship between the number of HTTP requests and query execution time, drawing on data collected across all experiments.

Subsequently, we assessed the resilience of our approach by gradually reducing the shape index information across the network by performing the following experiment:

[10] It should be noted that the type index and shape index approach are extensions of the LDP approach.

- We measured query execution time in networks where 0%, 20%, 50%, and 80% of Solid Pods expose a shape index.
- We measured query execution time in networks where 20%, 50%, and 80% shape indexes are complete.
- We measured query execution time in networks 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.

Results and Discussion

Evaluation Against Other Approaches

Analysis of the Query Execution Time Figure 5 demonstrates that the shape index approach performs queries faster than or comparable to state-of-the-art Solid Pod network traversal algorithms across all query templates except S4. In the appendix Figure 17 presents the ratio of execution time between the type index approach and the shape index approach. Table 3 presents the statistical significance by query templates. The shape index approach allows for successful termination of S7 template queries, which other approaches cannot complete due to timeouts resulting from an excessive number of HTTP requests. Query processing with the shape index approach requires as little as 13% of the execution time needed by the type index approach, achieving up to 7-fold performance improvement for S1 queries. 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. This table presents the average percentage of query-relevant resources per query template, derived from the where-provenance [13] of the query results and the number of HTTP requests. Queries with a high decrease of the number of HTTP request are known to be data-model selective due to the query-shape subsumption algorithm. In our context, a query that requests a single data model object is classified as data-model-selective, since in our benchmark the subwebs contain at most five distinct data model objects. From the analysis of the queries, those derived from templates D1, D4, S1, S3, and S7 (S3 is a template that cannot complete the execution before the timeout, while S7 can only be executed successfully with the shape index) are identified as data-model-selective and, as expected, exhibit a reduction in both the number of HTTP requests and the query execution time. Figure 6 shows the distribution of query execution times for each instance of the S1 query template. The distributions for all query templates are provided in the supplementary material. Queries from templates D6 and D7 show no reduction because they require nearly every data model objects in the datasets to be processed by the engine, making our approach ineffective in these cases. The D6 and D7 templates are presented in Listings 2 and 3. Figure 7 shows the distribution of query execution times of queries of the D7 templates. 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 as shown in Figure 8. The S4 template is presented in Listing 1. 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 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` [36], 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** and **H2** however the shape index approach can drastically increase the execution time when the structural properties are not used.

Temporal Analysis 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 [49, 47]. 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 [47]. 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 4 and 5 in the Appendix present the average results for the metrics presented above.

For Most Queries, the Shape Index Approach Tends to Produce the First Results Faster Figure 9 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 5, they still produced the first result faster than with the type index approach. This suggests that, although the shape index does

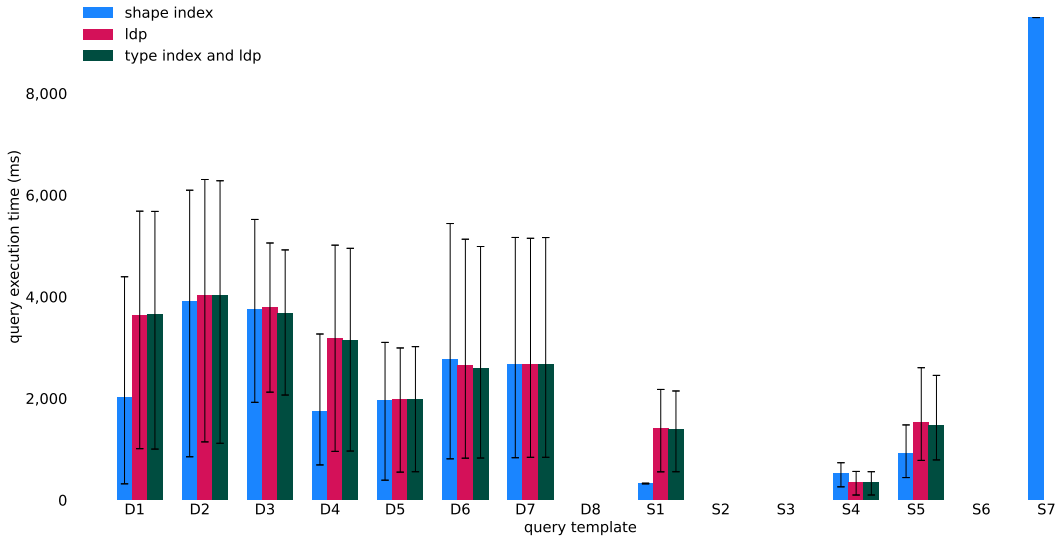

Figure 5. Performance comparison of different methods for query execution. The shape index performs better than both the type index and LDP for data-model selective queries. The shape index is also able to answer a query from template S7 which was not possible with the other approaches. For non-data-model selective queries, execution time differences are not statistically significant, as shown in Table 3. An exception occurs with queries of template S4, which do not leverage the structure of the publication environment. Results are presented by query template, leading to a multimodal distribution; therefore, variance analysis does not indicate statistical significance.

Approach/ Query Template	D1	D2	D3	D4	D5	D6	D7	S1	S4	S5
shape index (%)	$\mathbf{14^{42}_{6}}$	$\mathbf{38^{91}_{11}}$	$\mathbf{52^{62}_{37}}$	$\mathbf{15^{20}_{3}}$	$\mathbf{9^{13}_{2}}$	18^{41}_{7}	8^{11}_{2}	$\mathbf{12^{12}_{12}}$	6^{6}_{6}	$\mathbf{10^{11}_{4}}$
type index (%)	4^{6}_{4}	35^{86}_{8}	51^{61}_{36}	9^{15}_{3}	8^{12}_{2}	$\mathbf{20^{42}_{8}}$	$\mathbf{9^{13}_{2}}$	4^{8}_{1}	$\mathbf{80^{100}_{50}}$	3^{4}_{2}

Table 1. The percentage of query-relevant resources. 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.

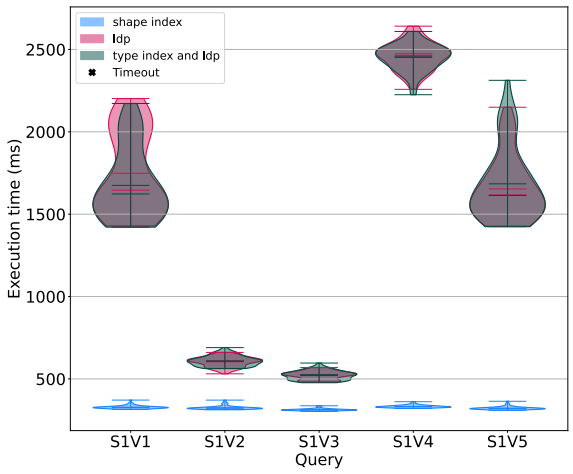

Figure 6. Distribution of query execution times (sample size: 50) for queries derived from the S1 template. The results show that the execution time of data-model selective queries is significantly reduced when using the shape index approach.

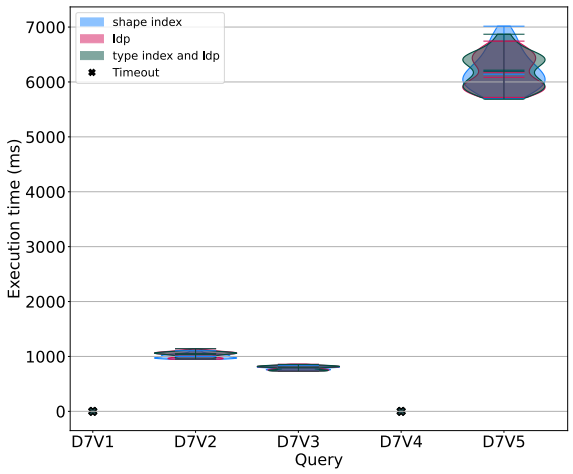

Figure 7. Distribution of query execution times (sample size: 50) for queries derived from the D7 template. The results show that the execution time of non-data-model selective queries is not significantly affected by the shape index approach.

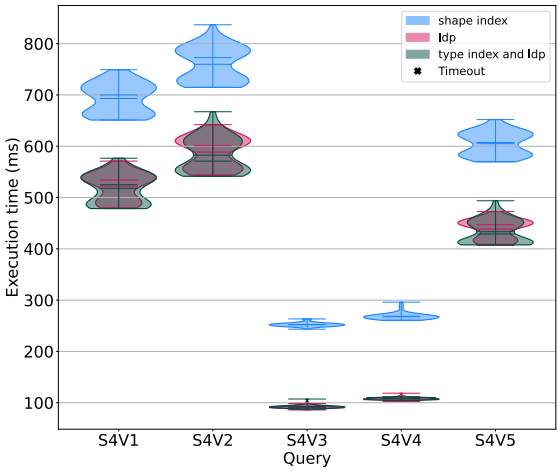

Figure 8. Distribution of query execution times (sample size: 50) for queries derived from the S4 template. Queries that do not exploit the structural properties of the dataset are significantly negatively impacted by the shape index approach.

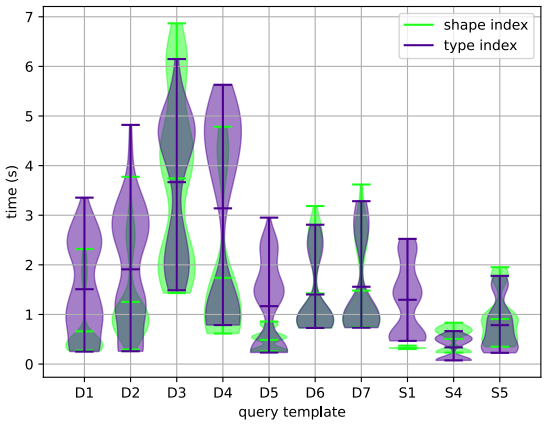

Figure 9. The first result distribution by query template. The shape index approach tends to produce the first results earlier than the type index approach, particularly for queries where the shape index achieves shorter total execution times.

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 9, 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.

The Shape Index Approach Can Reduce the Termination Time Figure 10 presents the termination times for the

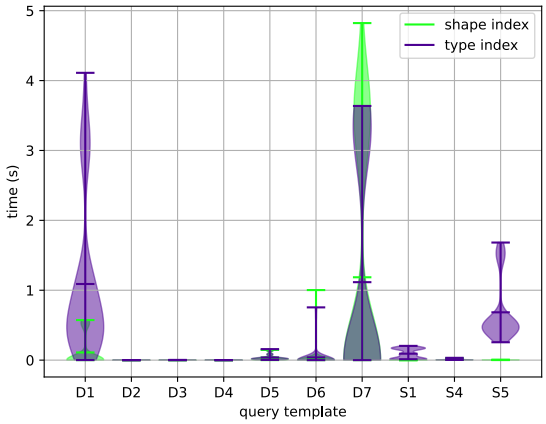

Figure 10. The termination time distribution by query template. With the exception of queries from templates D6 and D7, which have similar execution times, the shape index either maintains or improves query termination time.

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 5, an outlier query increases execution time by approximately a ratio of 1.2, which could explain the extended tail and the additional second of termination time.

The Shape Index Approach Generally Increases the Waiting Time Figure 11 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.

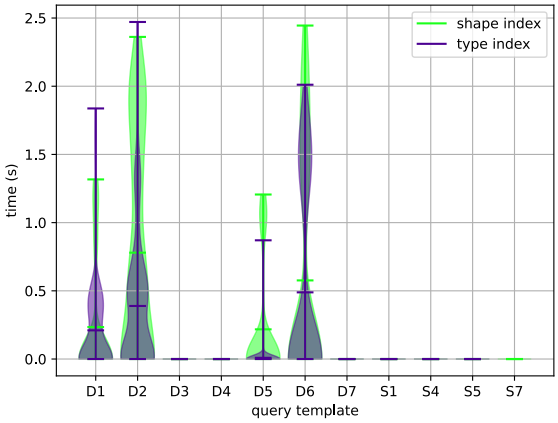

Figure 11. The waiting time distribution by query template. Except for queries from template D1, the shape index approach tends to either maintain or worsen the waiting time.

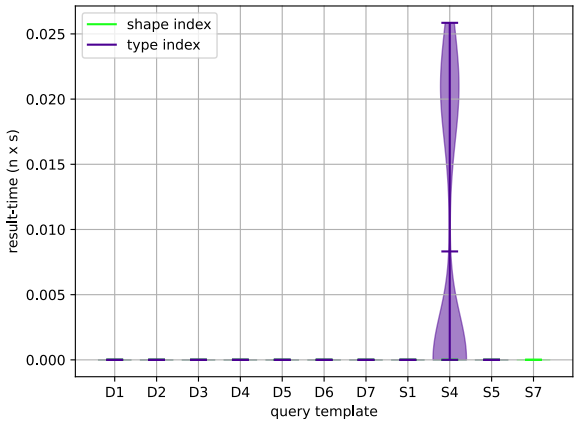

Figure 12. The *dief@0.1s* distribution by query template. At 0.1 second, the value is zero for every query template except S4. This is expected, as for all other templates the first result arrives after 0.1 seconds.

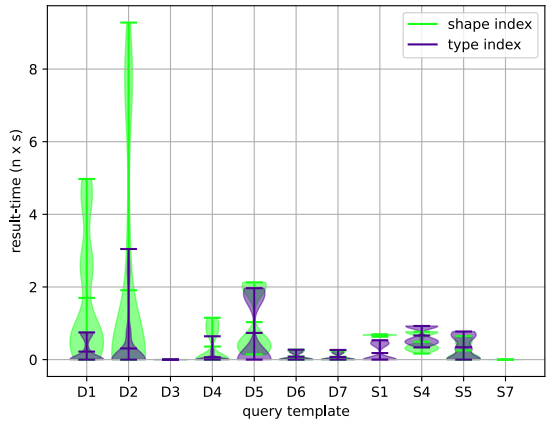

Figure 13. The *dief@1s* distribution by query template. At 1 second, the shape index approach shows a higher *dief@1s* (**higher is better**), indicating that more results can be obtained by users while maintaining a seamless flow of thought.

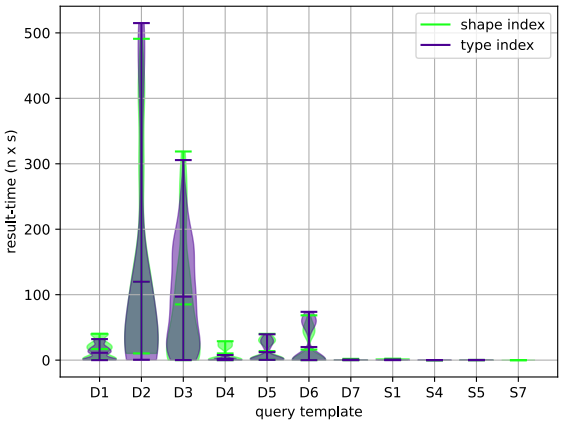

Figure 14. The *dief@lr* distribution by query template. At the time of the last result, the shape index approach tends to exhibit a similar *dief@t* (**higher is better**).

The Shape Index Approach Can Improve the Dieffi-ciency Metric Figures 12, 13, and 14 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*).[11] The *dief@0.1s* results show that, except for S4, no approach is able to produce instantaneous results, which aligns with the observations in Figure 9. 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 5 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. 11 These results suggest that, for fast early result arrival, the shape index approach tends to performs better. This finding is particularly relevant in the context of 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 compared to *dief@1s* (see Table 4 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 5ms (0.0039% of the timeout). Table 6 present these results. This outcome is expected, as the algorithm has polynomial time complexity, and the shapes

[11] 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.

and queries in the experiments are small and not deeply nested. This result validates **H3**.

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 15 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 5 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 templates 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, except for 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 performance. 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 to explain 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 **H4** and **H5** 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. **H6** is accepted; it is possible to retain performance gains even in networks with reduced shape-index information.

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 16 presents our analysis. The relationship between HTTP request and query execution time can be divided into two regimes. In the first regime (left 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 explanations for this behavior. First, the methods have some overhead due to the query-shape subsumption algorithm and the state retention of the pruning reachability criteria. However, the query-shape subsumption algorithm execution time is negligible for one execution and the state retention has a polynomial time complexity, thus it should not have a high impact on the execution time. Another explanation is the number of HTTP requests 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 (right 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,

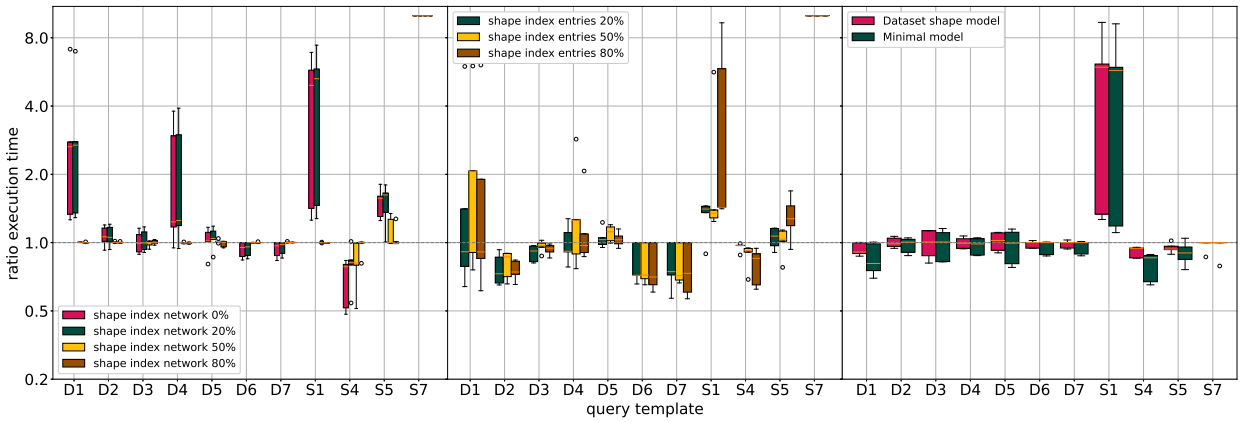

Figure 15. Shape index approaches tend to perform less effectively with limited network information and comparatively better where the baseline shape index underperforms. A higher ratio indicates a longer query execution time compared to a network with complete shape index information (**lower is better**).

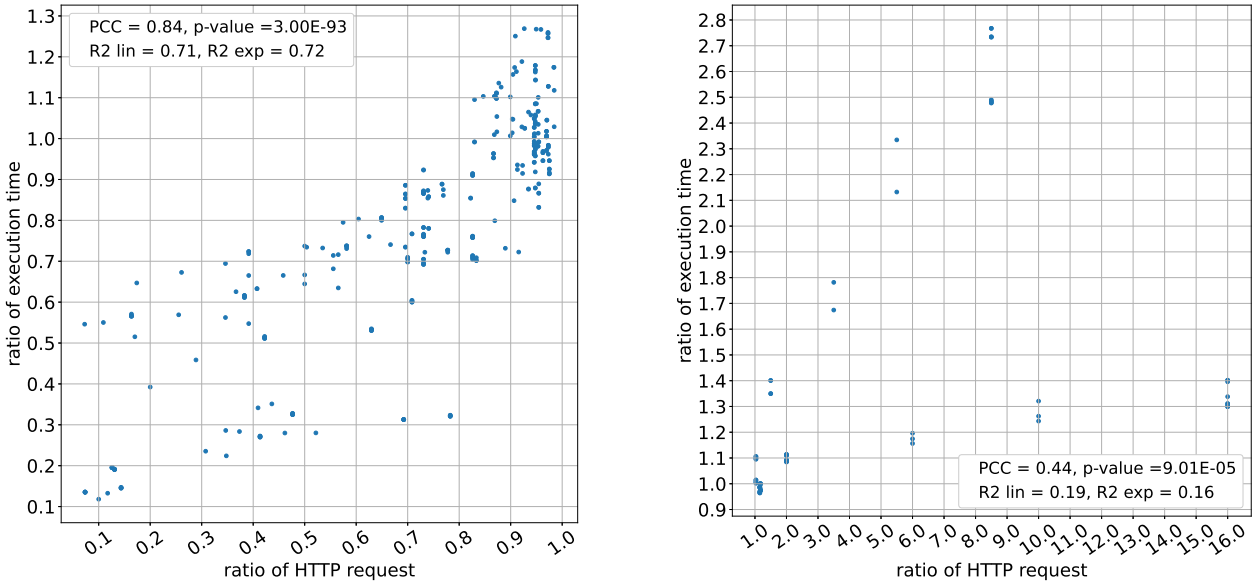

Figure 16. The data show two regimes in the relation between the number of HTTP requests and the execution time, we see a more linear correlation on the up figure than on the down figure.

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 **H7**, 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.

Summary of Findings

Across the evaluation, the shape index approach reduced both the number of retrieved sources and the query execution time for data-model selective queries while preserving result completeness, but it degraded performance for queries that do not exploit the structural properties of the network, such as those of template S4. The query-shape subsumption algorithm introduced negligible overhead. Reducing the amount of shape index information was not consistently detrimental: neither stricter constraints nor more complete indexes guaranteed better performance, yet the approach retained its gains even under partial coverage. Finally, the reduction in HTTP requests correlated with the decrease in execution time only when a sufficient number of requests were involved. Table 2 summarizes, for each hypothesis, whether the evaluation supports it.

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,

Hypothesis	Summary	Outcome
H1	Reduces sources and execution time for data-model selective queries	Supported
H2	No effect on non-data-model selective queries	Partially supported
H3	Negligible query-shape subsumption overhead	Supported
H4	Stricter shape constraints reduce HTTP requests more	Rejected
H5	More complete shape indexes give faster execution	Rejected
H6	Gains hold under partial shape-index coverage	Supported
H7	Linear HTTP-request / execution-time relationship	Partially supported

Table 2. Summary of the hypotheses and whether the evaluation supports them.

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 improved by a factor of 1.76. In the worst case, execution time increased by 2.8 times for a query that did not exploit the network structure; however, this query was already highly efficient, consuming only 0.30% of the maximum allowed execution time. 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. The construction and maintenance of shape indexes were beyond the scope of this work. They are lightweight metadata that is low-cost to produce and maintain, as descriptive shape indexes can be generated automatically [25] and their URI-template entries avoid exhaustive redefinition. This approach has the potential to benefit linked data-driven decentralization efforts such as the Solid ecosystem and support standardization efforts such as the Linked Web Storage Working Group [12] by improving query processing throughout these networks without requiring the maintenance of large centralized indexes between data providers. This work opens a new line of research in integrating data models directly into decentralized query execution, particularly for non-indexed networks. Future work could focus on supporting filter expressions in the query-subsumption algorithm, enhancing query planning in LTQP [61] 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. [13]

Acknowledgement

This research was supported by SolidLab Vlaanderen (Flemish Government, EWI RRF project VV023/10) and Serendipity Engine (Research Foundation - Flanders (FWO) grant number S006323N). Ruben Taelman is a postdoctoral fellow of the Research Foundation – Flanders (FWO) (1202124N).

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] T. Berners-Lee, R. Fielding, and L. Masinter. *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. https://datatracker.ietf.org/doc/html/rfc2396. [Accessed 11-06-2026]. Aug. 1998.

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

[9] 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.

[12] https://www.w3.org/groups/wg/lws/

[13] https://github.com/shapeIndexComunicaExperiment/documentation

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] M. Duerst and M. Suignard. *RFC 3987: Internationalized Resource Identifiers (IRIs)*. https://datatracker.ietf.org/doc/html/rfc3987. [Accessed 11-06-2026]. Jan. 2005.

[22] 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.

[23] Ruben Eschauzier, Ruben Taelman, and Ruben Verborgh. “Revisiting Link Prioritization for Efficient Traversal in Structured Decentralized Environments”. In: *Proceedings of the 24th International Semantic Web Conference*. Nov. 2025. URL: https://rubeneschauzier.github.io/r3-metric-data-processor/Revisiting_Link_Prioritization_for_Efficient_Traversal_in_Structured_Decentralized_Environments-camera-ready.pdf.

[24] 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/.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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].

[37] 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.

[38] 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/.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

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

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

[50] 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.

[51] 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.

[52] 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].

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

[54] 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.

[55] 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).

[56] 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.

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

[58] 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.

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

[60] 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.

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

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

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

[64] 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/.

[65] 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.

[66] 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.

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

[68] 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.

[69] Ruben Verborgh et al. "Web-Scale Querying through Linked Data Fragments." In: *LDOW*. 2014.

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

[71] 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.

[72] 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.

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

Appendix

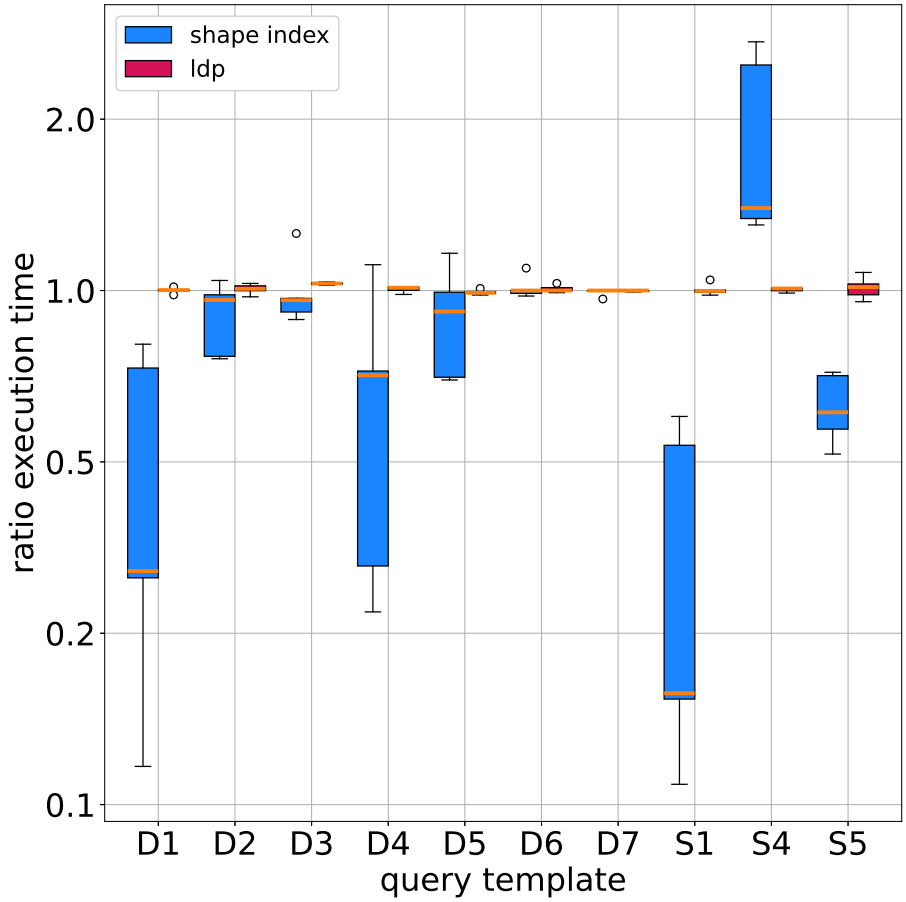

Figure 17. Comparison of query execution time with the type index approach. The ratio represents how the execution time of each method compares to that of the type index. A ratio greater than 1 indicates a slower execution time (**lower is better**). The shape index approach performs similarly or better than the other methods, except in the case of S4.

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 3. 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 4. 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 5. 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 6. Query-Shape containment computation time (100 samples) is negligeable with the most restrictive shapes of our experiments.

Template	Description	Data-model selective
D1	All posts created by a person, with their content and creation date.	Yes
D2	All messages (posts and comments) created by a person, with their content and creation date.	No
D3	The tags attached to a person's messages, ranked by how often each occurs.	No
D4	The locations of a person's comments, ranked by how often each occurs.	Yes
D5	The distinct IP addresses a person has sent messages from.	No
D6	The forums that contain a message created by a person.	No
D7	The moderators of the forums that contain a message created by a person.	No
S1	The profile of a person (name, gender, birthday, location, browser, creation date).	Yes
S3	The friends of a person and the date each friendship was created.	Yes
S4	The content (or image) and creation date of a message.	No
S5	The creator of a message.	No
S7	The comments that directly reply to a message, with their authors.	Yes

Table 7. Description of the query templates used in the evaluation. A *message* denotes either a post or a comment; the words *post* and *comment* are used when a template targets that type specifically.

Listing 1: The S4 (interactive-short-4) query template.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sn: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/data/>
PREFIX snvoc: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/vocabulary/>
PREFIX sntag: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/tag/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbpedia: <http://localhost:3000/dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://localhost:3000/dbpedia.org/ontology/>

SELECT
    ?messageCreationDate
    ?messageContent
WHERE
{
    ?message snvoc:id ?messageId .
    ?message snvoc:creationDate ?messageCreationDate .
    ?message snvoc:content|snvoc:imageFile ?messageContent .
}
```

Listing 2: The D6 (interactive-discover-6) query template.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sn: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/data/>
PREFIX snvoc: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/vocabulary/>
PREFIX sntag: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/tag/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbpedia: <http://localhost:3000/dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://localhost:3000/dbpedia.org/ontology/>

SELECT
DISTINCT
    ?forumId
    ?forumTitle
WHERE
{
    ?message snvoc:hasCreator ?person.
    ?forum snvoc:containerOf ?message;
            snvoc:id ?forumId;
            snvoc:title ?forumTitle.
}
```

Listing 3: The D7 (interactive-discover-7) query template.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sn: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/data/>
PREFIX snvoc: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/vocabulary/>
PREFIX sntag: <http://localhost:3000/www.ldbc.eu/ldbc_socialnet/1.0/tag/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbpedia: <http://localhost:3000/dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://localhost:3000/dbpedia.org/ontology/>

SELECT
DISTINCT
    ?firstName
    ?lastName
WHERE
{
    ?message snvoc:hasCreator ?person.
    ?forum snvoc:containerOf ?message;
        snvoc:hasModerator ?moderator.
    ?moderator snvoc:firstName ?firstName .
    ?moderator snvoc:lastName ?lastName .
}
```