

How many and what type of SPARQL queries can be answered through zero-knowledge link traversal?

Pavlos Fafalios

L3S Research Center, Leibniz University of
Hannover, Germany

fafalios@L3S.de

Yannis Tzitzikas

Computer Science Department, University of Crete, and
Information Systems Laboratory, FORTH-ICS, Greece

tzitzik@ics.forth.gr



Outline

- Introduction / Motivation
- Problem description
- **Finding Linked-Data answerable queries**
- **Executing Linked-Data answerable queries**
- **Experimental Results**
- Conclusion and Future Directions

Introduction / Motivation

The Web of Data

- The **Linked Open Data Cloud**
 - 1,239 datasets with 16,147 links (Mar'19)
- Data Markup
 - JSON-LD - <https://json-ld.org/>
 - RDFa - <https://rdfa.info/>
- How to query this data?
 - SPARQL

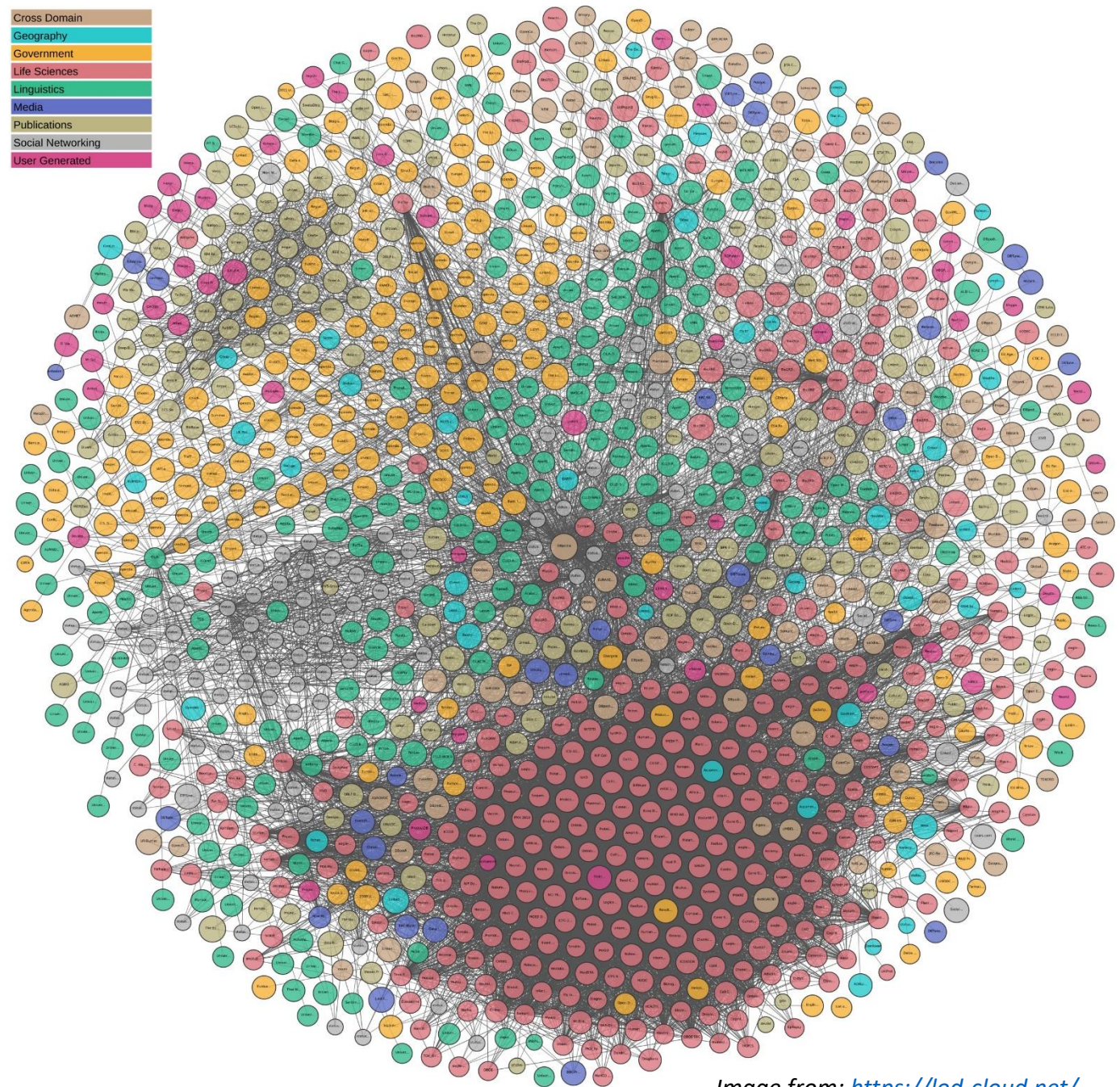


Image from: <https://lod-cloud.net/>

SPARQL endpoints

- Web services that accept SPARQL queries and return results in a machine-readable format
 - SPARQL endpoint of DBpedia: <http://dbpedia.org/sparql>

```
PREFIX dbr: <http://dbpedia.org/resource/>  
PREFIX dbo: <http://dbpedia.org/ontology/>  
  
SELECT ?birthDate WHERE {  
  dbr:Donald_Trump dbo:birthDate ?birthDate }
```



"1946-6-14^^<http://www.w3.org/2001/XMLSchema#date>

- Limitations
 - Low reliability
 - Not optimized for efficiency
 - Limited bandwidth
 - Expensive to host and maintain
- **Need for alternative, less demanding query evaluation methods!**

Approaches to query Web data

I. Data centralization

- Provide a query service over a collection of RDF data (gathered from different sources)
 - Then query the data through SPARQL
 - Fast responses, but **no fresh results**
 - **Cost for maintaining a centralized repository**
- **Linked Data Fragments (LDFs):** efficient offloading of SPARQL query execution from servers to clients
 - A LDF is a resource consisting of triples that match a specific selector, together with metadata and hypermedia controls
 - Triple Pattern Fragment (TPF): triple pattern as *selector*, count *metadata*, *controls* to retrieve other TPFs
 - Servers maintain high availability rates
 - **Requires the setup and maintenance of dedicated servers and clients**



Approaches to query Web data

II. Query federation

- Integrated access to distributed RDF sources on the Web
- DARQ, SemWIQ: two of the first systems to support the execution of queries to multiple endpoints
- SPARQL 1.1 Federated Query
 - SERVICE operator
- **Requires the remote data to be available through SPARQL endpoints**
- SPARQL-LD^[1]:
 - Extension of SERVICE operator to enable querying any web source containing RDF data
 - HTML pages (RDFa, JSON-LD), RDF files (N3, RDF/XML, Turtle), Web APIs (returning RDF data), ...

```
SELECT ?birthDate WHERE {  
  ?person rdf:type :Politician .  
  SERVICE <http://dbpedia.org/sparql>  
    ?person dbo:birthDate ?birthdate }
```

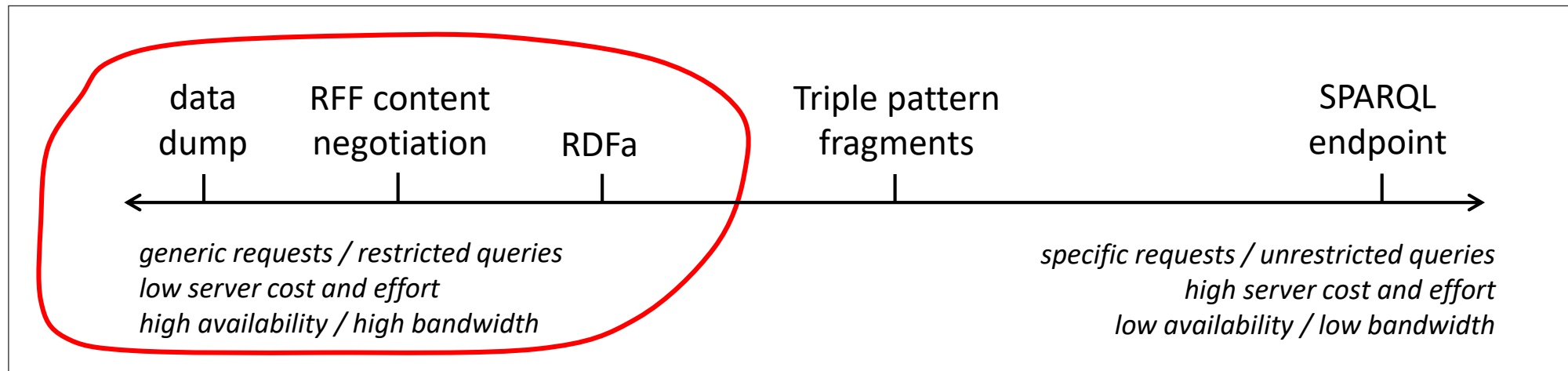
```
SELECT ?paperTitle WHERE {  
  SERVICE <http://l3s.de/~fafalios/> {  
    ?paper a swrc:Paper ; dc:title ?paperTitle } }
```

[1] P. Fafalios and Y. Tzitzikas, "SPARQL-LD: A SPARQL Extension for Fetching and Querying Linked Data", ISWC'15

Approaches to query Web data

III. Link Traversal

- Real time URI lookup / dynamically discover data relevant for answering a query (by following RDF links)
 - No access to local or remote endpoints
 - Relies on Linked Data principles, thus on robust web protocols (HTTP, URI)
-
- **Zero-knowledge link traversal**
 - No starting graph, seed URIs, or pre-build indexes for starting the link traversal



This figure is a variation of the figure at: <http://linkeddatafragments.org/concept/>

Our focus: zero-knowledge link traversal

- What types of SPARQL queries can be **directly** executed on the **live Web of Linked Data**, without a priori knowledge of available data sources?
 - Input: just a SPARQL query
 - Starting point: the URI(s) that exist in the query
 - Additional URIs are resolved only if this is needed for satisfying a triple pattern (for binding its variables)
- Why?
 - **Convenience**: avoid setting up and maintaining indexes/servers/endpoints
 - **Freshness of results**: in line with the dynamic nature of the Web
 - **Decentralisation**: motivates publishers to put their data online (e.g., by just uploading RDF files)
 - **Reliability**: relies on robust web protocols (HTTP, URI)

Contributions

- (A) Method for finding **Link Data-answerable Queries (LDAQ)**
- (B) Method for transforming LDAQ to SPARQL-LD queries
- (C) Experimental results (using real SPARQL query logs)
 - Pattern-based analysis of LDAQ and non-LDAQ
 - Efficiency of the transformed queries

- ❖ Open source: <https://github.com/fafalios/LDAQ>
 - + Results of pattern-based analysis of LDAQ and non-LDAQ

Problem Description

Query evaluation through zero-knowledge link traversal (Example 1)

- Find the birth date of Barack Obama:
 - Run the following query at <http://dbpedia.org/sparql> (SPARQL endpoint of Dbpedia)

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?birthDate WHERE {
  dbr:Barack_Obama dbo:birthDate ?birthDate }
```

- Through *zero-knowledge link traversal*:
 - STEP 1: Access the URI http://dbpedia.org/resource/Barack_Obama and retrieve all triples
 - STEP 2: Run the triple pattern (dbr:Barack_Obama dbo:birthDate ?birthDate) on the retrieved triples

Query evaluation through zero-knowledge link traversal (Example 1)

- Implementation through **SPARQL-LD**:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?birthDate WHERE {
  dbr:Barack_Obama dbo:birthDate ?birthDate }
```



```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?birthDate WHERE {
  SERVICE dbr:Barack_Obama {
    dbr:Barack_Obama dbo:birthDate ?birthDate } }
```

Query evaluation through zero-knowledge link traversal (Example 2)

- Find the birth date of all basketball players in DBpedia
- Run the following query at <http://dbpedia.org/sparql> (SPARQL endpoint of Dbpedia)

```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?player ?birthDate WHERE {
  ?player a dbo:BasketballPlayer .
  ?player dbo:birthDate ?birthDate }
```

- Through zero-knowledge link traversal:
 - STEP 1: Access the URI <http://dbpedia.org/ontology/BasketballPlayer> (class) and retrieve all triples
 - STEP 2: Run the triple pattern (?player a dbo:BasketballPlayer) on the retrieved triples (i.e., find instances of the class dbo:BasketballPlayer)
 - STEP 3: Access the URIs of all instances of the class dbo:BasketballPlayer (URI-bindings of variable ?player) and retrieve all triples
 - STEP 4: Run the triple pattern (?player dbo:birthDate ?birthDate) on the retrieved triples of each instance

Query evaluation through zero-knowledge link traversal (Example 2)

- Implementation through **SPARQL-LD**:

```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?player ?birthDate WHERE {
  ?player a dbo:BasketballPlayer .
  ?player dbo:birthDate ?birthDate }
```



```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?player ?birthDate WHERE {
  SERVICE dbo:BasketballPlayer {
    ?player rdf:type dbo:BasketballPlayer }
  SERVICE ?player {
    ?player dbo:birthDate ?birthDate } }
```

Query evaluation through zero-knowledge link traversal (Example 3)

- Find the total number of triples

```
SELECT count(*) WHERE {  
  ?s ?p ?o }
```



- Find all subjects having the name “Barack Obama”

```
SELECT ?entity WHERE {  
  ?entity foaf:name “Barack Obama” }
```



Requirements (in line with the Linked Data principles)



Rule 1

- URIs must be dereferenceable and return RDF data



Rule 2

- URIs must provide both the incoming and outgoing properties of the corresponding resource (all triples where the URI is the subject or object)
 - This includes URIs that represent RDFS/OWL classes, meaning that the URI of a class should return all its instances

(A) Finding Linked-Data answerable queries

(A) Finding Linked Data-answerable Queries (LDaQ)

- Algorithm for checking the answerability of a BGP:
 1. Go through the triple patterns and find “bindable” variables, i.e.:
 - variables that can get bound by dereferencing a subject or object URI, or
 - variables that can get bound through bindings of other variables
 2. Map a non-bindable variable to other variables that can help binding it at a later stage
 3. At the end, if there is at least one variable that is not bound or cannot get bound through another bound variable, the query is not a LDaQ

```
?player a dbo:BasketballPlayer .  
?player dbo:birthplace ?place .  
?place rdfs:label ?placeName
```

- *What if we change the order of the triples?*

```
?player dbo:birthplace ?place .  
?player a dbo:BasketballPlayer .  
?place rdfs:label ?placeName
```

- *What if there is no subject or object URI?*

```
?player dbo:birthplace ?place .  
?place rdfs:label ?placeName
```

(A) Finding Linked Data-answerable Queries (LDaQ)

- Algorithm for checking the answerability of a query:
 1. Go through the **triple** and **UNION group** elements
 2. Check the answerability of an element using the previous algorithm and considering the already bound variables
 - In case of UNION group, check the answerability of each UNION's graph pattern
 - If the element is not answerable, add it to a list of "pending" elements (it might be answerable when a variable in another element gets bound)
 - At each step, update the list of bound variable
 3. Go through the pending elements and check for bindable variables iteratively until the list is empty
 - In each loop, at least one new element must get answerable, otherwise the query is not a LDaQ

```
SELECT ?player ?birthDate ?birthPlaceName WHERE {  
  { ?player rdf:type dbo:BasketballPlayer } UNION { ?player rdf:type dbo:FootballPlayer }  
  ?player dbo:birthDate ?birthDate .  
  ?player dbo:birthPlace ?place .  
  { ?place foaf:name ?birthPlaceName } UNION { ?place rdfs:label ?birthPlaceName } }
```

(B) Transforming Linked-Data answerable queries

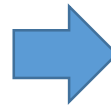
(B) Transforming LDaQ to SPARQL-LD queries

- Objective: Evaluate a SPARQL query over the live Web of Linked Data
 - *without accessing local or remote endpoints*
 - *without considering any seed URIs*
- Approach: Transformation-based method through SPARQL-LD
 - Directly make use of this functionality through existing instances of SPARQL-LD
 - No need to implement a dedicated server!

(B) Transforming LDaQ to SPARQL-LD queries

- Algorithm for transforming a BGP to a SPARQL-LD graph pattern
 1. Go through the triples and create SERVICE patterns
 2. If the triple contains a URI or bound variable, check if there is already a SERVICE pattern for it
 - If so, add the triple to its graph pattern, otherwise create a new SERVICE pattern
 3. If the triple does not contain a URI or a bound variable, add it to a list of “pending” triples
 - Since the BGP is Linked Data-answerable, these triples require the binding of another variable existing in a subsequent triple
 4. Go through the pending triples iteratively until the list is empty
 - In each loop, either include the triple to an existing SERVICE pattern or create a new one.

```
?player a dbo:BasketballPlayer .  
?player dbo:birthPlace ?place .  
?place rdfs:label ?placeName
```



```
SERVICE dbo:BasketballPlayer {  
  ?player a dbo:BasketballPlayer }  
SERVICE ?player {  
  ?player dbo:birthPlace ?place }  
SERVICE ?place {  
  ?place rdfs:label ?placeName }
```

(B) Transforming LDaQ to SPARQL-LD queries

- Algorithm for transforming a query to a SPARQL-LD graph pattern
 1. Go through all elements (triples or UNION groups) and check if they are Linked Data-answerable
 2. If so, include the corresponding element to the SPARQL-LD query, either by appending it to an existing SERVICE or by creating a new one
 3. If not, add the element to a list of “pending” elements (whose transformation requires the binding of a variable existing in as subsequent triple or UNION group)
 4. Go through the pending elements iteratively until the list is empty
 - In each loop, include the transformed element to the SPARQL-LD query

(B) Transforming LDaQ to SPARQL-LD queries

- Algorithm for transforming a query to a SPARQL-LD graph pattern

```
SELECT ?player ?birthDate ?birthPlaceName WHERE {  
  { ?player rdf:type dbo:BasketballPlayer } UNION { ?player rdf:type dbo:FootballPlayer }  
  ?player dbo:birthDate ?birthDate .  
  ?player dbo:birthPlace ?place .  
  { ?place foaf:name ?birthPlaceName } UNION { ?place rdfs:label ?birthPlaceName } }
```



```
SELECT ?player ?birthDate ?birthPlaceName WHERE {  
  { SERVICE dbo:BasketballPlayer { ?player rdf:type dbo:BasketballPlayer } } UNION  
  { SERVICE dbo:FootballPlayer { ?player rdf:type dbo:FootballPlayer } }  
  SERVICE ?player {  
    ?player dbo:birthDate ?birthDate ; dbo:birthPlace ?place }  
  SERVICE ?place {  
    { ?place foaf:name ?birthPlaceName } UNION { ?place rdfs:label ?birthPlaceName } } }
```

(B) Transforming LDaQ to SPARQL-LD queries



- Problems
 - Dereferencing a URI may result in the retrieval of an unforeseeable large set of RDF triples
 - Servers might put restrictions on clients such as serving only a limited number of requests per second
 - A link traversal-based query execution system should implement a politeness policy
 - E.g., by respecting the robots.txt
- We have not (yet) examined (and implemented) the following SPARQL operators:
 - DESCRIBE
 - look up the URI and return all triples
 - FROM, FROM NAMED / GRAPH
 - Look up the URI, fetch the triples, and run the graph pattern over these triples
 - SERVICE (over remote endpoints)
 - Just check if the graph pattern is Linked Data-answerable

(C) Experimental Results

Experiments - Objectives

- Find patterns of LDaQ and non-LDaQ
 - Check their number, type, and distribution
- Examine the efficiency of the transformed SPARQL-LD queries

Experiments - Datasets



Dataset	#Queries	#Invalid	#Unconsidered	#Remaining	#Unique
LGD	4,240,736	456,393	1,148,809	2,635,534	670,809
SWDF	13,990,138	224,849	3,326,767	10,438,522	789,049
BM	129,989	0	0	129,989	129,989
BIO2RDF	192,057	47	2	192,008	62,819
DBPEDIA	49,296,201	2,003,381	3,869,723	43,423,097	16,028,271
67,849,121					

- Fixing of common errors (like absence of popular prefixes)
- Jena 3.2 to parse the queries
- We did not consider queries that:
 - are not valid according to Jena 3.2
 - Use property paths, nested queries, or contain one of the following (unexamined) operators: DESCRIBE, FROM, GRAPH, SERVICE, MINUS, EXISTS, BIND, VALUES

Pattern-based analysis of LDaQ and non-LDaQ

- Pattern (template) extraction

- Remove FILTER
- Replace:
 - Variable → [V], URI → [U], Literal → [L], Blank node → [B]
 - UNION → UN

```
SELECT ?player ?birthDate ?birthPlaceName WHERE {  
  { ?player rdf:type dbo:BasketballPlayer } UNION { ?player rdf:type dbo:FootballPlayer }  
  ?player dbo:birthDate ?birthDate ;  
    dbo:birthPlace ?place .  
  { ?place foaf:name ?birthPlaceName } UNION { ?place rdfs:label ?birthPlaceName } }
```



```
{ [V] [U] [U] } UN { [V] [U] [U] } [V] [U] [V] ; [U] [V] { [V] [U] [V] } UN { [V] [U] [V] }
```

Pattern-based analysis of LDaQ and non-LDaQ

Number of answerable and not answerable queries and unique patterns

Dataset	#Test queries	#LDaQ	#LDaQ patterns	#Non-LDaQ	#non-LDaQ patterns
LGD	670,809	572,720 (85.4%)	444	98,089 (14.6%)	197
SWDF	789,049	677,923 (85.9%)	570	111,126 (14.1%)	202
BM	129,989	129,936 (99.9%)	4	53 (0.04%)	1
BIO2RDF	62,819	60,740 (96.7%)	9	2,079 (3.30%)	5
DBPEDIA	16,028,271	14,053,584 (87.7%)	2,816	1,974,687 (12.3%)	780



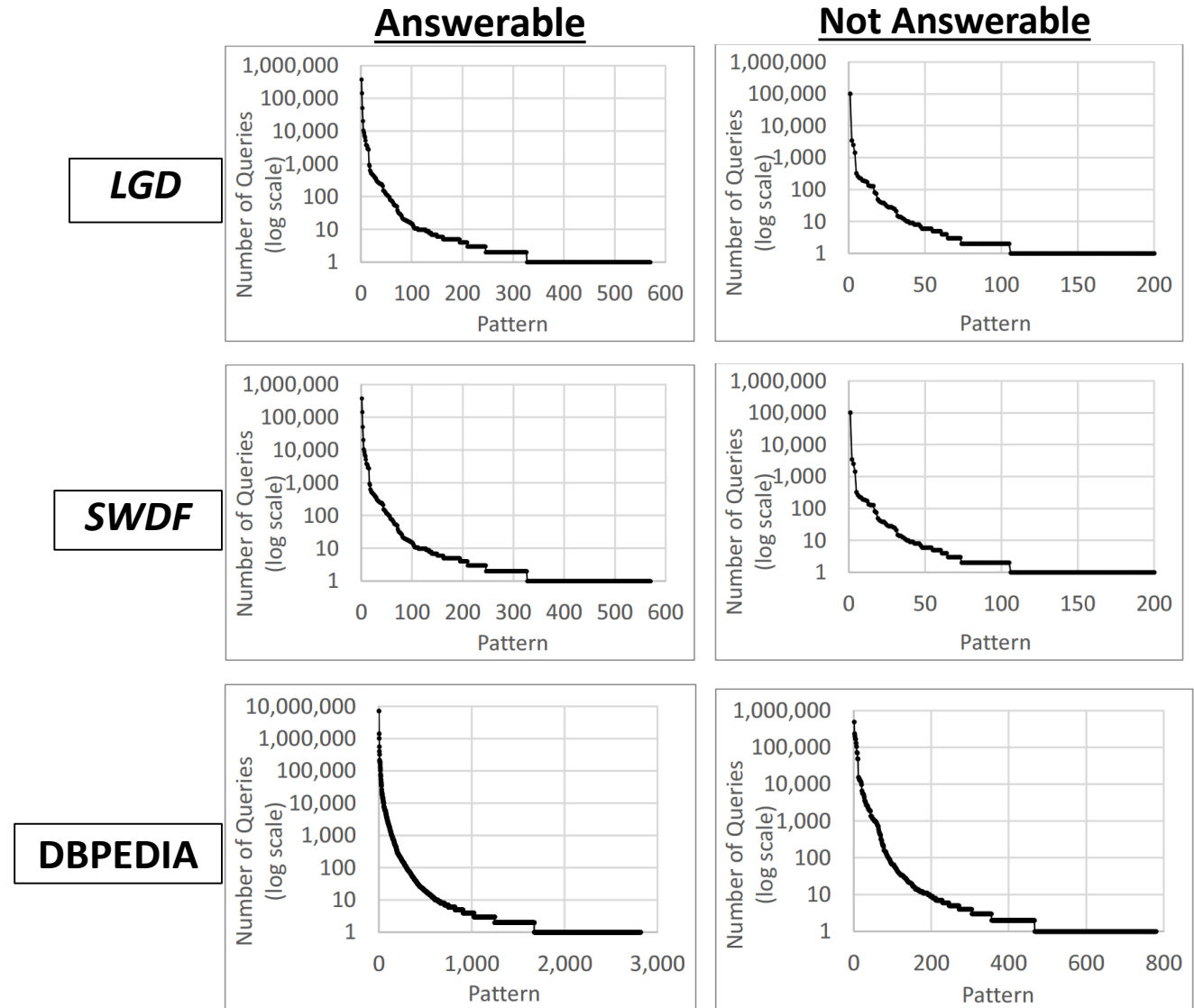
More than 85% of the examined queries are potentially Linked Data-answerable!

Distribution of LDaQ and non-LDaQ patterns

- Power law distribution
- Top-10 LDaQ patterns:
 - LGD: 95%
 - SWDF: 95%
 - Dbpedia: 84%
- Top-10 non-LDaQ patterns:
 - LGD: 98%
 - SWDF: 96%
 - Dbpedia: 86%



The majority (>84%) of both answerable and non-answerable queries follow a few (≤ 10) specific patterns/templates!



Top LDaQ and non-LDaQ patterns

LGD

1 [U] [V] [V]
2 [V] [V] [U]
3 OPT { [U] [U] [V] }
4 [V] [U] [U] ; [V] [V]
5 [V] [U] [U]

Answerable

1 [V] [U] [V]
2 [V] [U] [V] . [V] [U] [V]
3 [V] [U] [L]
4 [V] [U] [L] OPT { [V] [U] [V] } OPT { [V] [U] [V] }
5 [V] [V] [V]

Not Answerable

SWDF

1 [U] [U] [V]
2 [V] [U] [U]
3 [U] [V] [V]
4 { [U] [V] [V] } UN { [V] [V] [U] }
5 [U] [V] [V] OPT { [U] [U] [V] }

Answerable

1 [V] [U] [L]
2 [V] [U] [V]
3 [V] [U] [L] ; [U] [V] . [V] [U] [V]
4 OPT { [V] [U] [V] }
5 [V] [U] [V] ; [U] [V]

Not Answerable

DBPEDIA

1 [U] [U] [V]
2 { [U] [U] [U] } UN { [U] [U] [U] }
3 [V] [U] [U] ; [U] [L] . [V] [U] [U] { [V] [U] [V] } UN { [V] [U] [V] } UN {
[V] [U] [V] } UN { [V] [U] [V] } { [V] [U] [V] } UN { [V] [U] [V] } OPT {
[V] [U] [V] } OPT { [V] [U] [V] } OPT { [V] [U] [V] }
4 [U] [V] [V]
5 { [V] [U] [U] } UN { [V] [U] [U] } [V] [U] [L] . [V] [U] [V] ; [U] [L] ; [U] [V]

Answerable

1 { [V] [U] [L] } UN { [U] [U] [V] } [V] [U] [V] ; [U] [V] . [V] [U] [V]
2 [V] [U] [L] ; [V] [V] OPT { [V] [U] [V] }
3 { [V] [U] [L] } UN { [V] [U] [V] ; [U] [L] } UN { [V] [U] [V] ; [U] [L] } OPT {
[V] [U] [V] } OPT { [V] [U] [V] ; [U] [V] } OPT { [V] [U] [V] } OPT { [V] [U]
[V] } OPT { [V] [U] [V] } OPT { [V] [U] [V] } OPT { [V] [U] [V] }
4 [V] [U] [L]
5 [V] [V] [V] . [V] [U] [L]

Not Answerable

Efficiency of the transformed queries


- Querying a single URI
 - Patterns like [U] [V] [V] and [V] [U] [U]
 - 70% of all unique queries in LGD
 - 77% of all unique queries in SWDF
 - 56% of all unique queries in DBpedia
- Time proportional to the number of triples contained in the resource [1]
 - 10,000 triples → ≈1 second
 - 1M triples → ≈30 seconds
 - Querying Dbpedia URIs:
 - ≈320 ms (N3)
 - ≈650 ms (content negotiation)
 - ≈300 ms (endpoint)
- Potential problem when requesting the incoming properties of resources representing classes
 - 3.6% of queries in DBpedia
 - The number of instances can be very high, e.g., dbo:Person has 3.2M instances



More than 50% of the examined DBpedia queries can bypass the endpoint and be efficiently (<1 sec) answered through link traversal!

[1] Fafalios, P., Yannakis, T., & Tzitzikas, Y. (2016). Querying the Web of Data with SPARQL-LD. In *International Conference on Theory and Practice of Digital Libraries* (pp. 175-187). Springer.

Efficiency of the transformed queries

- Querying multiple URIs
 - The majority of queries containing joins
- Experiments for the pattern [V] [U] [U] ; [U] [V] 
- Query execution time highly depends on number of intermediate bindings
- We tested the following (Wikicat) classes:
 - a) American Civil Rights Lawyers (136 instances)
 - b) Video Artists (262 instances)
 - c) People from Sheffield (502 instances)
 - d) American magazines (1,030 instances)
 - e) American Male Film Actors (9,787 instances)
- Two different query execution methods
 - Non-optimized (sequential fetching of remote resources)
 - Optimized (fetching using max 10 parallel threads at the same time)

```
?player a dbo:BasketballPlayer .  
?player foaf:name ?name
```

Efficiency of the transformed queries

- Query execution time (seconds) of the transformed queries

	(a)	(b)	(c)	(d)	(e)
Non-optimised	26	44	79	152	1,322
Optimised	7	13	24	48	423

- (a) American Civil Rights Lawyers (136 instances)
- (b) Video Artists (262 instances)
- (c) People from Sheffield (502 instances)
- (d) American magazines (1,030 instances)
- (e) American Male Film Actors (9,787 instances)



For queries with large number of intermediate bindings (which in turn might require large number of URI lookups), the query execution time can become prohibitively high!

Conclusion and Future Directions

Conclusion and Takeaways

- Answering SPARQL queries through **zero-knowledge link traversal**
 - *Detecting answerable queries*
 - *Transforming answerable queries to SPARQL-LD queries (that bypass the endpoint)*
- Popular patterns of answerable and non-answerable queries?
- Efficiency of the transformed queries?



More than 85% of the (examined) queries are potentially Linked Data-answerable!



The majority (>84%) of both answerable and non-answerable queries follow a few (≤ 10) specific patterns/templates!



More than 50% of the examined DBpedia queries can bypass the endpoint and be efficiently (<1 sec) answered through link traversal!



For queries with large number of intermediate bindings (which in turn might require large number of URI lookups), the query execution time can become prohibitively high!

Future directions



- Long-term vision:
 - Decrease the load of SPARQL endpoints and increase their availability/reliability!
- Design adaptive query processing methods that combine different strategies
 - Based on: the load of the servers/endpoints, the availability of the remote sources, and the estimated efficiency of query execution
- Study methods to improve the execution time of the transformed SPARQL-LD queries
 - Caching? Query planning?
- Further examination of non-answerable query patterns
 - Would a different policy for publishing Linked Data be beneficial for making more queries answerable through Link Traversal?

Thank you!

Questions / Comments?

fafalios@L3S.de

- Finding and transforming **LDaQ**: <https://github.com/fafalios/LDaQ>
- **SPARQL-LD**: <https://github.com/fafalios/sparql-ld>

