UNIVERSITÄT KOBLENZ · LANDAU

WeST People and Knowledge Networks

Benchmarks for SPARQL Property Paths

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B. Sc.)
im Studiengang Informatik

vorgelegt von

Adrian Skubella

Erstgutachter: Prof. Dr. Steffen Staab
Institute for Web Science and Technologies

Zweitgutachter: M. Sc. Daniel Janke
Institute for Web Science and Technologies

Koblenz, im Juli 2016

Hiermit gestatte ich, dass meine Abschlussarbeit in der Bibliothek eingestellt und unter der Creative Commons Lizenz (CC BY-SA 3.0 DE)[1] im Internet veröffentlicht wird. Darüber hinaus gestatte ich, dass mein Quellcode unter der Apache Lizenz 2.0[2] im Internet veröffentlicht werden darf.

_____	_____
Ort/Datum	Unterschrift

1 https://creativecommons.org/licenses/by-sa/3.0/de/legalcode
Kurzfassung: https://creativecommons.org/licenses/by-sa/3.0/de/

2 https://www.apache.org/licenses/LICENSE-2.0

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Koblenz, den 15.07.2016 _____

Abstract

The Resource Description Framework (RDF) is a triple based representation of directed graphs with labelled edges. With the emergence of RDF graphs special databases, called RDF stores, were developed. In order to query graphs, which are stored in these RDF stores, the query language SPARQL Protocol And Query Language (SPARQL) is used. With the help of this query language it is possible to describe subgraphs with a fixed number of edges, containing the desired information. In march 2013 SPARQL 1.1 was released. The newly introduced property paths make it possible to describe subgraphs with an arbitrary amount of edges. This feature was implemented into the different RDF stores. In order to compare the implementations of property paths of the different RDF stores a set of metrics is needed. Benchmarks can provide such metrics. The development of such a benchmark and the comparison of different RDF stores based on this benchmark are the substance of this bachelor thesis.

Zusammenfassung

Mit dem Aufkommen von Resource Description Framework (RDF) Graphen - eine trippelbasierte Repräsentation von gerichteten, kantengelabelten Graphen - sind auch spezielle Graphdatenbanken die RDF Stores entwickelt worden. Um die in diesen Stores gespeicherten Graphen anfragen zu können, wird die Anfragesprache SPARQL Protocol And RDF Query Language (SPARQL) verwendet. Mit ihrer Hilfe lassen sich Teilgraphen mit einer festen Anzahl von Kanten beschreiben, die die gesuchten Informationen enthalten. Im März 2013 ist SPARQL 1.1 veröffentlicht worden. Die neu eingeführten Propery Paths erlauben es nun, Teilgraphen mit einer variablen Kantenzahl zu beschreiben. Dieses Feature wurde in den vergangenen Jahren von den RDF Stores implementiert. Um die Property Path Implementierungen der verschiedenen RDF Stores miteinander vergleichen zu können, werden verschiedene Metriken benötigt. Benchmarks stellen solche Metriken zur Verfügung. Die Entwicklung eines solchen Benchmark und der Vergleich der verschiedenen RDF Stores basierend auf diesem Benchmark geschieht in dieser Bachelorarbeit.

Contents

1 Introduction

Linked Data is structured data published and connected through the internet. A growing number of data providers provide their data in the linked open data cloud. This data is represented in the Resource Description Framework (RDF) format. The RDF is a triplebased representation of directed, labelled graphs. In order to store these graphs, different RDF stores have been developed. In order to query the data in the RDF stores the SPARQL Protocol And RDF Query Language (SPARQL) is used.

In 2013 SPARQL 1.1 was introduced which included Property Paths. Property Paths make it possible to traverse an arbitrary amount of edges in a dataset. For instance sequences of different edges or alternative edges can be traversed. It is also possible to traverse all edges that connect different nodes directly or indirectly with each other. In practice this can be used to find all friends of a friend of a person in a social network. Another example is to find all subtypes of a class in an graph. For instance all different subtypes of animals and their subtypes could be returned from a database holding information about animals.

With the introduction of Property Paths in 2013, already existing RDF stores have been adapted to support Property Paths. Now the question arises, how well these adapted RDF stores support Property Paths. In order to compare the efficiency of different RDF stores benchmarks can be used. Therefore, benchmarks measure certain performance metrics of databases enabling the comparison based on the measured metrics. Thus, with the help of benchmarks it can be decided, which RDF store delivers better results with respect to the chosen performance metrics. In order to investigate how well different RDF stores support Property Paths a benchmark has been developed in the context of this work, which evaluates this feature.

Research Questions

This bachelor thesis has two main research questions:

Research question 1: How can the support of Property Paths be benchmarked?

1.1 What are quality criteria for benchmarks?

1.2 How well do existing RDF store benchmarks fulfil these quality criteria?

The benchmark should test to what extent RDF stores support Property Paths and if complete and sound results for queries with Property Paths are returned by the stores. Furthermore the efficiency of the RDF stores should be tested in terms of the execution time of queries, which contain Property Paths.

Research question 2: How efficiently do different RDF stores process queries containing Property Path expressions?

In order to answer research question 2, 4 different RDF stores have been benchmarked and the results have been compared.

Methodology

In order to develop a benchmark for SPARQL Property Paths an introduction to RDF, RDF stores, SPARQL and Property Paths is given first, to provide the basic principles of this work. After that a literature search on quality criteria for good benchmarks has been done, in order to define own criteria, which then have been used to evaluate existing RDF store benchmarks. This evaluation has been summarized and discussed to point out what can be

learned from existing RDF store benchmarks.

Based on the summary of the evaluation of existing benchmarks it has been discussed how the elements of the new benchmark have to be designed, in order to test how efficiently RDF stores support Property Paths. Then, the new Property Path Benchmark has been evaluated in order to show that it fulfils the prior defined quality criteria.

In order to answer research question 2 and therefore, show how well different RDF stores handle Property Paths, the benchmark has been used to benchmark 4 different RDF stores. The results of the benchmark have been evaluated and compared to stress to what extent the tested stores support Property Paths. Finally the results of this work have been summarized in the last part of this thesis and possible future research is presented.

2 Preliminaries

In this section an introduction to RDF, RDF stores, and the query language for the RDF SPARQL will be given in order to provide the basis for understanding this thesis.

2.1 Resource Description Framework

The Resource Description Framework is a general-purpose language for representing information in the web[1]. It uses triples to represent the information as directed, labelled graphs. An excerpt of a fictitious social network with the name social-nw is shown in listing 1. It is written in the n-triple format, which is used for representing RDF data[2]. The *Internationalized Resource Identifier* (IRI) `http://www.social-nw.com/Steve` in line 1 labels the subject node. The IRI `http://www.social-nw.com/person/feature#knows` in line 2 denotes the predicate and the IRI `http://www.social-nw.com/Ann` in line 3 is the object. The first part of each triple is a subject and the last part is an object. Both are labelled nodes and denote the start and the end node of an edge. The second part is a predicate which denotes the label of the edge connecting these nodes. The graphical representation of the triple is shown in figure 1.

```
1 <http://www.social-nw.com/Steve>
2 <http://www.social-nw.com/person/feature#knows>
3 <http://www.social-nw.com/Ann>.
```

Listing 1: Steve knows Ann relationship as an RDF triple

Figure 1: Graphical representation of an RDF triple

The label identifying a node is an IRI or in case of an object it can be a literal, too. The predicate itself is an IRI and denotes a property of the subject. Furthermore, a label identifying a node can be a blank node. Blank nodes are treated as indicator for the existence of an anonymous node, without using an IRI for identification. Even though, blank nodes

[1] https://www.w3.org/standards/techs/rdf#w3c_all retrieved at 27.01.16
[2] https://www.w3.org/TR/n-triples/ retrieved at 7.03.16

are part of RDF they are neglected in the context of this work, because they are not of major importance for this work.

Definition 2.1. RDF triple
The triple $(s, p, o) \in I \times I \times (I \cup L)$ is called RDF triple where I and L are two disjoint sets of all IRIs and literals, respectively. Furthermore, s is called the subject, p the predicate and o the object of the triple.[1]

As stated in definition 2.1, the object of triples can be an IRI or a literal. In figure 2 three RDF triples with IRIs as well as literals as object are shown. `"Steve Miller"` is a literal, shown by the rectangular shape of the node, whereas all other nodes are IRIs. A set RDF triples as shown in figure 2 is called an RDF graph.

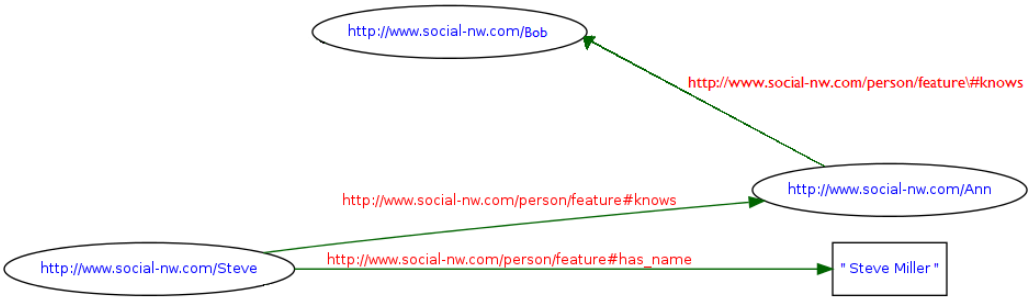

Figure 2: An RDF graph with a literal

Definition 2.2. RDF graph
An RDF graph G is a finite set of RDF triples.[2]

In graphs paths can be traversed from one node to another in order to retrieve the desired information from a graph.

Definition 2.3. Path
A path $P = (< v_1, e_1, v_2 >, < v_2, e_2, v_3 >, ..., < v_n, e_n, v_{n+1} >)$ in an RDF graph G is a sequence of triples such that $\forall i, j : i \neq j \implies v_i \neq v_j$ and $(v_i, e_i, v_{i+1}) \in G$. [2]

An example for a path from Steve to Bob in the graph shown in figure 2 is shown in listing 2.

```
1 (<http://www.social-nw.com/Steve,
2 http://www.social-nw.com/person/feature#knows,
3 http://www.social-nw.com/Ann>,
4 <http://www.social-nw.com/Ann,
5 http://www.social-nw.com/person/feature#knows,
6 http://www.social-nw.com/Bob>)
```

Listing 2: A path from Steve to Bob

2.2 SPARQL

In order to query data from an RDF graph the SPARQL Protocol And RDF Query Language is used. For the RDF graph shown in figure 2 a simple query, retrieving all persons Steve

knows, is shown in listing 3. In this query the namespaces `snw` and `snwf` in line 1 and 2 are used to abbreviate the IRIs `http://www.social-nw.com/` and `http://www.social-nw.com/person/feature#` for better legibility.

```
1 PREFIX  snw:  <http://www.social-nw.com/>
2 PREFIX  snwf: <http://www.social-nw.com/person/feature#>
3
4 SELECT ?friend WHERE{
5          snw:Steve snwf:knows ?friend
6          }
```

Listing 3: A SPARQL query retrieving all friends of Steve

Definition 2.4. Basic graph pattern (BGP)
A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a basic graph pattern, where *I, L* and *V* are disjoint sets of IRIs, literals and variables respectively.
If P_1 and P_2 are basic graph patterns, then $(P_1\ .\ P_2)$is a graph pattern.[2]

These basic graph patterns can be used with SELECT queries of SPARQL to retrieve data from an RDF graph.

Definition 2.5. SELECT query
If P is a graph pattern and V is a set of variables, then (`SELECT` V `WHERE` P) is a SELECT query.[1]

In line 4 of listing 3 the `SELECT` statement followed by the variable `?friend` and `WHERE` denotes that only values matching the variable `?friend` for the BGP in line 5 are returned. The semantics of graph patterns are defined in terms of mappings, which are defined in definition 2.6

Definition 2.6. Variable mappings
Mappings are partial functions from variables V to an RDF term T, which is defined as $I \cup L$.[3]

The domain $dom(\mu)$ of a mapping μ is the set of variables on which μ is defined. Two mappings can be compatible as defined in definition 2.7.

Definition 2.7. Compatible mappings
Two mappings μ_1 and μ_2 are compatible (written as $\mu_1 \sim \mu_2$) if $\mu_1(x) = \mu_2(x)$ for all variables x that are in both $dom(\mu_1)$ and $dom(\mu_2)$.[3]

If $\mu_1 \sim \mu_2$, then $\mu_1 \cup \mu_2$ denotes the mapping obtained by extending μ_1 according to μ_2 on all variables in $dom(\mu_1)$ and $dom(\mu_2)$. The join of two sets of mappings is defined in definition 2.8.

Definition 2.8. Join of mappings
Given two sets of mappings M1 and M2, the join of M1 and M2 is defined as:
$M_1 \bowtie M_2 \implies \{\mu_1 \cup \mu_2 | \mu_1 \in M_1, \mu_2 \in M_2, \wedge \mu_1 \sim \mu_2\}$ [3]

For a triple pattern P and a mapping μ, $\mu(P)$ is written for the triple obtained from P by replacing each variable $x \in dom(\mu)$ by $\mu(x)$. In the following definition the evaluation $[[P]]_G$ of a graph pattern P over a graph G is defined. The set of all variables appearing in a pattern P is denoted by var(P).

Definition 2.9. Evaluation of graph pattern
if $P \in (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$, then $[[P]]_G := \{\mu : var(P) \to T | \mu(P) \in G\}$,
if P is $P_1.P_2$, then $[[P]]_G := [[P_1]]_G \bowtie [[P_2]]_G$ [3]

Finally the evaluation of `SELECT` queries is defined in definition 2.10.

Definition 2.10. Semantics of SELECT query
The evaluation $[[Q]]_G$ of a query Q of the form `SELECT X WHERE P` is the set of all projections $\mu|_X$ of mappings μ from $[[P]]_G$ to X, where the projection of $\mu|_X$ is the mapping that coincides with μ on X and is undefined elsewhere.[3]

Property Paths

In 2013, SPARQL 1.1 was introduced. With the new version of SPARQL *Property Paths* were introduced.

Definition 2.11. Property Paths
sPo is a Property Path where $s \in V \cup I$, $o \in I \cup L \cup V$ and P is a Property Path expression.

There are several different Property Path expressions, which denote different paths. These different expressions and their syntax are presented in definition 2.12.

Definition 2.12. Property Path expression
1) $p \in I$ is a Property Path expression.
2) $\hat{}P$ with Property Path expression P, is the inverse Property Path expression.
3) P_1/P_2, with Property Path expressions P_1 and P_2, is the sequence Property Path expression.
4) $P_1|P_2$, with Property Path expressions P_1 and P_2, is the alternative Property Path expression.
5) $P*$, with Property Path expression P, is the transitive reflexive closure Property Path expression.
6) $P+$, with Property Path expression P, is the transitive closure Property Path expression.

The evaluation of a Property Path is presented in the following definition:

Definition 2.13. Evaluation of Property Paths
For constants $s \in I, o \in I \cup L$ and variables $v, v_1, v_2 \in V$ the evaluation of Property Paths is defined as:

$$[[sPo]]_G := \begin{cases} \{\mu = \varnothing\} | if(s,o) \in [[P]]_G \\ \varnothing, otherwise \end{cases}$$

$$[[vPo]]_G := \{\mu | (\mu(v), o) \in [[P]]_G \wedge dom(\mu) = \{v\}\}$$
$$[[sPv]]_G := \{\mu | (s, \mu(v)) \in [[P]]_G \wedge dom(\mu) = \{v\}\}$$
$$[[v_1 P v_2]]_G := \{\mu | (\mu(v_1), \mu(v_2)) \in [[P]]_G \wedge dom(\mu) = \{v_1, v_2\}\}$$

The semantics of the evaluation of Property Path expressions are defined in definition 2.14.

Definition 2.14. Evaluation of Property Path expressions
The evaluation $[[P]]_G$ of a Property Path expression P over an RDF graph G is a set of pairs of RDF terms from $I \cup L \cup V$ defined as follows:
$[[p]]_G := \{(s,o) \mid (s,p,o) \in G\}$,
$[[\hat{}P]]_G := \{(s,o) \mid (o,s) \in [[P]]_G\}$,
$[[P_1/P_2]]_G := [[P_1]]_G \circ [[P_2]]_G$,
$[[P_1|P_2]]_G := [[P_1]]_G \cup [[P_2]]_G$,
$[[P+]]_G := \bigcup_{i>1}[[P^i]]_G$,
$[[P*]]_G := [[P+]]_G \cup \{(s,s) \mid (s,p,o) \in G\}$,
where $\circ$ is the usual composition of binary relations, and p^i is the concatenation $p/.../p$ of i copies of p.[3]

In some cases Property Paths allow a more concise expression of some SPARQL basic graph patterns. Furthermore, Property Paths can be used to express the transitive closure as described in definition 2.12. For instance, instead of describing all paths directly when retrieving all friends of a friend of a person in a social network the transitive reflexive closure operator * can be used as shown in listing 4 [4].

```
1 PREFIX  snw:  <http://www.social-nw.com/>
2 PREFIX  snwf: <http://www.social-nw.com/person/feature#>
3
4 SELECT ?foaf WHERE{
5          snw:Steve snwf:knows* ?foaf
6          }
```

Listing 4: Example of a query containing a Property Path expression retrieving all friends of a friend of Steve

In listing 4 a SPARQL query containing a Property Path is shown. In line 5 the predicate `snwf:knows` followed by `*` retrieves all nodes which have a direct or indirect knows relationship to Steve including Steve himself. This means that Steve, all friends of Steve and all friends of a friend of Steve are looked up. Then they are returned, as denoted by `SELECT ?foaf` in line 4. Executing this query on the RDF graph shown in figure 2 would return `snw:Steve, snw:Ann` and `snw:Bob`.
The traversal length describes the exact amount of edges that are traversed in the actual data graph. The supremum and infimum are defined as follows:

Definition 2.15. Traversal Length Supremum
$$tl^G_{Sup}(p) = \begin{cases} 1, if\ |[[p]]_G| > 0, \\ 0, otherwise \end{cases}$$
$tl^G_{Sup}(\hat{}P) = tl^G_{Sup}(P)$
$tl^G_{Sup}(P1/P2) = tl^G_{Sup}(P1) + tl^G_{Sup}(P2)$
$tl^G_{Sup}(P1|P2) = max(tl^G_{Sup}(P1), tl^G_{Sup}(P2))$
$tl^G_{Sup}(P+) = tl^G_{Sup}(P) + tl^G_{Sup}(P*)$
$tl^G_{Sup}(P*) = max(\{n \mid \bigcup_{i=0}^{n-1}[[P^i]]_G \neq \bigcup_{i=0}^{n}[[P^i]]_G\}) * tl^G_{Sup}(P)$

Definition 2.16. Traversal Length Infimum

$$
\begin{aligned}
tl^G_{Inf}(p) &= \begin{cases} 1, if\ |[[p]]_G| > 0, \\ 0, otherwise \end{cases} \\
tl^G_{Inf}(\hat{}P) &= tl^G_{Inf}(P) \\
tl^G_{Inf}(P1/P2) &= tl^G_{Inf}(P_1) + tl^G_{Inf}(P_2) \\
tl^G_{Inf}(P1|P2) &= min(tl^G_{Inf}(P_1), tl^G_{Inf}(P_2)) \\
tl^G_{Inf}(P+) &= 1 \\
tl^G_{Inf}(P*) &= 0
\end{aligned}
$$

Query Graphs

SPARQL queries can be displayed as query graphs [5]. In accordance to the query, three query graph structures are possible:

1. Star shaped query graphs have a central vertex and several edges connected to this central vertex as created by the query in listing 5 and illustrated in figure 3.[5]

```
1 PREFIX  snw:  <http://www.social-nw.com/>
2 PREFIX  snwf: <http://www.social-nw.com/person/feature#>
3
4 SELECT ?foaf ?hobby ?age ?wife WHERE{
5           snw:Steve snwf:knows ?foaf.
6           snw:Steve snwf:likes ?hobby.
7           snw:Steve snwf:age ?age.
8           snw:Steve snwf:isMarried ?wife.
9           }
```

Listing 5: Example of a query creating a star shaped query graph

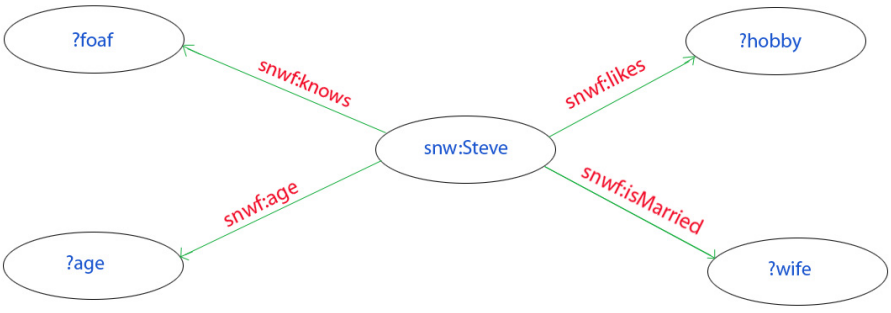

Figure 3: A star shaped query graph

2. Chain shaped query graphs consist of a single path as created by the query in listing 6 and illustrated in figure 4.[5]

```
PREFIX  snw:  <http://www.social-nw.com/>
PREFIX  snwf: <http://www.social-nw.com/person/feature#>

SELECT ?age WHERE{
        snw:Steve snwf:isMarried ?wife.
        ?wife snwf:age ?age.
        }
```

Listing 6: Example of query creating a chain shaped query graph

Figure 4: A chain shaped query graph

3. Complex shaped query graphs are also called hybrid shaped query graphs. It is the mixture of a star shaped and a chain shaped query graph as created by the query in listing 7 and illustrated in figure 5.[5]

```
PREFIX  snw:  <http://www.social-nw.com/>
PREFIX  snwf: <http://www.social-nw.com/person/feature#>

SELECT ?age ?hobby ?homeTown ?firstName ?company WHERE{
        snw:Steve snwf:knows ?foaf.
        ?foaf snwf:age ?age.
        ?foaf snwf:isFrom ?homeTown
        snw:Steve snwf:isMarried ?wife.
        ?wife snwf:hobby ?hobby.
        ?wife snwf:firstName ?firstName
        ?wife snwf:works ?company
        }
```

Listing 7: Example of a query creating a complex query graph

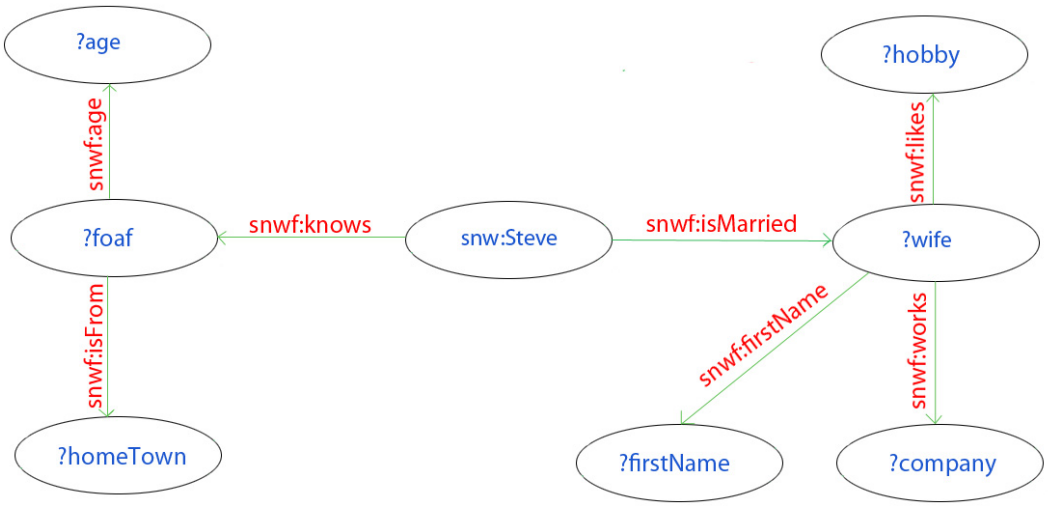

Figure 5: A complex query graph

3 Existing Benchmarks

After the basics of RDF and SPARQL have been given, the existing benchmarks can be examined. Therefore, the individual elements of the benchmarks are identified first. Then, quality criteria for the whole benchmark as well as for the individual benchmark elements are defined in order to evaluate the existing benchmarks. After that the evaluation of the benchmarks based on the quality criteria is given. Finally the results of the evaluation are summarized in order to provide a basis for the design of the new Property Path Benchmark.

3.1 Elements of a Benchmark

In order to process queries on an RDF graph an RDF store, i.e., a graph database for RDF graphs, is usually used. Different stores that support SPARQL Property Paths exist, for instance Apache Jena[3] , Virtuoso[4] and Allegrograph[5]. In order to compare how well the different RDF stores adapted Property Paths performance metrics of benchmarks can be used. These benchmarks usually consist of 4 parts:

Dataset. One part of a benchmark is a dataset. The dataset may come with a data generator to create a synthetic dataset with, or it consists of real data taken from a real dataset. This dataset has to be stored and will then be queried.

Queries. A set of queries that will be executed on the given dataset.

Execution strategy. The execution strategy dictates the way the benchmark run is executed. Among others it determines how many times given queries are executed and how metrics are measured.

[3]https://jena.apache.org/ retrieved at 13.12.2015
[4]http://virtuoso.openlinksw.com/ retrieved at 13.12.2015
[5]http://franz.com/agraph/allegrograph/ retrieved at 13.12.2015

Metrics. A set of metrics which makes it possible to compare different data stores with each other.

3.2 Quality Criteria for Benchmarks

In [6] Huppler defines several characteristics a benchmark for any system has to fulfil in order to be a good benchmark.

Relevant. The results should reflect important aspects of a tested system.

Repeatable. The benchmark should reproduce the same results if it is executed a second time on the same system.

Fair. The benchmark does not prefer one system in particular.

Verifiable. The results have to be comprehensible.

Economical. The test sponsors can afford to run the benchmark.

Huppler stresses that not all of these aspects have to be fulfilled. Furthermore, it is stated that in order to fulfil the last four characteristics the first one could not be fulfilled to full extend. For instance in order to fulfil the economical characteristic and make the benchmark affordable the relevance might be neglected and therefore compromises have to be done.[6]

In [7] Gray defines 4 main criteria for good benchmarks. The benchmark should be

Relevant. The benchmark should measure the performance of a system.

Portable. The benchmark should be easy to use on different systems.

Scalable. The benchmark should apply to small and large computer systems. Therefore it should be possible to scale the benchmark up to larger systems.

Simple. The results of the benchmark should be easily readable and understandable.

The following quality criteria are partly based on Huppler‘s and Gray‘s quality criteria for benchmarks. But mainly the criteria are adapted from [8]:

Goal. A benchmark should define a goal, which describes what exactly the benchmark wants to evaluate.

Continuity. This criterion says that the benchmark should evolve with the progress made in the field it is used in.

Syntactical correctness. The datasets and test cases should be syntactical correct. This means, the data should be expressed in a universal coding in order to avoid parse errors. Furthermore, queries should be error-free, accompanied by a natural language description, and finally expressed in a standard query language, e.g., SPARQL 1.1.

Independence. The benchmark should not benefit from one specific system but should be driven only by the tasks to be solved. Due to the fact that examining this aspect for each benchmark would go beyond the scope of this work, it will only be analysed to which extend it is possible that a benchmark is specialised for a certain RDF store. An example for an independent dataset would be a dataset where the user can choose the properties of the dataset for himself. Queries, which especially test technological challenges could be independent queries.

Scalability. The size of the used datasets should scale in order to test different systems.

Representativity. The used datasets and queries should mimic real world data and queries.

Dissemination. The benchmark should be publicly and freely available.

Intelligibility. It should be determined how the resulting data is represented in order to make sure it is easy to be read and understood by everyone.

Diversity. The benchmark should be able to identify different advantages of different systems. Aspects, which make this possible are:

- The query graph structures should vary.
- The diversity of metrics. This means that a benchmark should not just measure one aspect of a tested system like the execution time of queries, but at least two aspects.

Criticality. In order to show the limitations of a system different critical queries should be tested within the benchmark. Critical queries are queries with a high traversal length. A query has a sufficient traversal length if it is at least 4.

It has to be stressed that the representativity and the scalability of the dataset of a benchmark are in competition to some degree. Due to the fact that a real dataset is representative but not scalable and a synthetic, scalable dataset is not representative compromises have to be done as stated in [6] and [7]. These criteria are used in the following sections to evaluate existing benchmarks for RDF stores.

3.3 Berlin SPARQL Benchmark

The Berlin SPARQL Benchmark (BSBM) was designed along an e-commerce use case and a SPARQL and SQL version is available[6]. The latest version 3.1 was released 26.08.11.

Dataset. The dataset of the BSBM is based on an e-commerce use case, in which different vendors offer a set of products. Furthermore, different users have reviewed these articles and posted their review on different review sites. The dataset consists of different classes. For instance, the class product. Product has several properties, e.g., a label, a product type, a comment and a producer. The product type itself has properties too like a label and a comment and a property defining whose sub class it is. In this way all nodes in the graph are connected. The dataset is generated with a data generator and can be scaled with the amount of products as scale factor. Besides the plain RDF representation of the dataset, it also can be represented in a named graph data model or in a relational data model.

[6] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/ retrieved at 20.01.16

Queries. The queries are summarized in three sets of queries called query mixes. A query mix consist of 25 queries which are designed to emulate the search and navigation pattern of a consumer looking for products. It consists of 8 steps which are close to a real life scenario, e.g. searching for a product, specifying the search, retrieving reviews and offers for a product. These query mixes enable the measurement of the performance of an RDF store against different use cases. The queries are available for each of the three data models of the dataset. [7]

Execution strategy. The BSBM contains a test driver which makes it possible to alter different variables defining the execution strategy. The default execution strategy is to execute 10 query mixes without measuring any metrics to warm up the store. The number of query mixes used can be changed manually. After that a use case has to be chosen to determine which queries are executed. Then 50 query mixes are executed by default and the metrics are measured. The number of query mixes can be changed manually. Finally, the results are returned and the benchmark execution is finished.

Metrics. The BSBM focuses on three metrics:

- Queries per second: Average amount of queries that were executed per second.
- Query mixes per hour: Number of query mixes with different parameters that were executed per hour.
- Overall runtime: Overall time it took to execute a certain amount of query mixes.

There are more metrics for single queries like the average query execution time or for queries mixes the composite query execution time. A list of all metrics can be found at `http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/BenchmarkRules/index.html#metrics`. Thus, BSBM focuses on metrics for measuring the execution time of queries but also provides a metrics called price/performance metric. This metrics describes how good a system is in respect to the financial investment.

Evaluation

The following can be said in regard to the different quality criteria mentioned in section 3.2:

Goal. The BSBM is designed around an e-commerce in order to test how well a store can handle realistic queries.

Continuity. From 2008 when the first version of the BSBM was released until 2011 when the latest version 3.1 was released the BSBM was regularly updated. But since BSBM has not been maintained since 2011 and the BSBM does not consider features of SPARQL 1.1 the continuity criterion is not fulfilled.

Syntactical correctness. The dataset is represented in three different formats: The n-triple format for RDF, the TriG syntax for the named graph and SQL for the relational schema[8]. The queries are accompanied by a natural language description and error-free. Furthermore, they are available in SPARQL 1.0.

Independence. The BSBM defines a use case around which the benchmark is designed. The synthetic dataset and queries are designed close to this use case.

[7] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/V1/spec/index.html#queries retrieved at 20.01.16

[8] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/Dataset/index.html#relationalrepresentation retrieved at 18.02.16

Scalability. The dataset can be scaled by an arbitrary factor. This factor determines the amount of products in the dataset.

Representativity. The queries are chosen close to the use case scenario and represent a generic search for a product so that the queries are close to real world queries. The dataset represents products, their producer, reviews etc. Thus, this data is close to real life data but has the limitation to be completely synthetic. A problem which arises from this is that a very high structuredness is guaranteed in this dataset. The structuredness is the probability with which the instances of a class have every property defined by the class. In this case a dataset where every product has a label, a product type, a comment and a producer would have a very high structuredness. A dataset consisting of products which often miss these properties would have a low structuredness. According to a study by Duan et. al. the structuredness of synthetic RDF data, commonly used for benchmarks, is very high, whereas the structuredness of real RDF data is rather low[9].

Dissemination. The BSBM is freely available and can be used for academic and industrial users.

Intelligibility. On the one hand the BSBM offers a full disclosure report in xml containing all information of the benchmark execution. On the other hand it offers single results in a readable format. The benchmark results are named according to the scenario, the scale factor of the dataset and the number of concurrent clients. For instance, `23.7 QPS(2)-NTS(10000,1))` means that on average 23.7 queries of type 2 were executed per second by a single client stream against a native triple store containing data about 10,000 products.

Diversity. The queries of BSBM have either a star shaped query graph or a complex query graph. A plain chain shaped query graph is missing. Even though BSBM focuses on several metrics for measuring the execution time of queries it also provides metrics for estimating the quality of the system in respect to the financial investment.

Criticality. The queries look up information for a product or look up products of certain types. Due to the fact, that the dataset is highly structured the queries can retrieve answers after traversing at most 2 edges. Thus the traversal length is low and the criticality criterion is not fulfilled.

3.4 The SPARQL Performance Benchmark

The SPARQL Performance Benchmark (SP^2Bench) is based on the Digital Bibliography and Library Project (DBLP), which was developed by the Albert-Ludwigs-Universität Freiburg[9]. It executes fixed queries on real world RDF data that is synthetically scaled.

Dataset. The dataset of the SP^2Bench is based on the DBLP, an online accessible bibliographical collection of scientific publications and releases hosted by the university of Trier[10]. The data generator supports the generation of arbitrarily large DBLP-like data in NTriples syntax. It supports data generation up to a given triple count or up to a given file size[11]. The dataset generation is deterministic and therefore the identical dataset is created when the generation process is executed with the same parameters independent from operating systems or machines. The structure of the dataset is based on a study of the structure of the DBLP data. In [10] it is claimed

[9] http://dbis.informatik.uni-freiburg.de/forschung/projekte/SP2B/ retrieved at 22.01.16
[10] http://dblp.uni-trier.de/ retrieved at 22.01.16
[11] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/data.php retrieved at 23.01.16

that by this study several structural constrains were designed, which then can be used to design representative data.

Queries. The query set consists of 14 `SELECT` and 3 `ASK` queries. It is stated that the queries vary in characteristics like selectivity, query and output size, and different types of joins. On the bases of this characteristics queries were designed which start from simple queries with a low traversal length and go on to more complex queries with higher traversal lengths.[12]

Execution strategy. Even though no default execution strategy is mentioned in [10], several existing RDF stores were benchmarked with SP^{2}Bench. The execution strategy for those stores was to execute the queries for datasets of different sizes. First for a dataset of 10k triples, then 50k, 250k, 1M, 5M, and finally 25M triples. For each size the queries were executed 3 times without warming up the store. The resulting metrics were then presented for each store.

Metrics. The SP^{2}Bench has 5 metrics that are described:

- Success rate: The rate with which queries succeed. A query fails if any error occurs during its execution or a timeout occurs after 30 minutes.
- Loading time: In [10] it is stated that the user himself should report on the loading times for the documents of different sizes. The metric applies to systems with a database backend. It can be ignored for in-memory engines because there the loading is typically part of the evaluation process.
- Per-Query performance: The individual performance results for each query consisting of its execution time and if it was a successfully executed as defined in Success Rate.
- Global Performance: This metric takes the execution time of each query into consideration and should evaluate the overall performance of a system. If q_i is query number i and $t(q)$ is the execution time of a query then the global performance is defined as: $\sqrt[17]{\prod_{i=1}^{17} t(q_i)}$
  Failed queries have an execution time of 3600 seconds as a penalty.
- Memory consumption: This metric measures the maximum memory consumption of queries and the average memory consumption of all queries.[10]

Evaluation

To which extend SP^{2}Bench fulfils the quality criteria mentioned in section 3.2 is presented in the following list.

Goal. SP^{2}Bench aims to focus on a variety of metrics and not only the execution time. For instance also the memory consumption or time needed for loading data into the store is considered.

Continuity. Due to the fact, that the last update of SP^{2}Bench was in 2009 and SPARQL 1.1 was introduced in 2013 the continuity is not given for the benchmark.

Syntactical correctness. The dataset of SP^{2}Bench is highly structured, error-free and written in the n-triple format. Furthermore, the queries are error-free to and accompanied by a natural language description[13]. The queries are all written in SPARQL 1.0.

[12] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/queries.php retrieved at 22.01.16
[13] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/queries.php 16.01.16

Independence. The dataset of Sp^{2}Bench is synthetic and therefore not necessarily independent. Furthermore, the queries were chosen in order to test technological challenges. These challenges are independent from any RDF store in particular and thus, the queries are independent.

Scalability. The data generator of SP^{2}Bench is both scalable and performant such that the size of the dataset can be chosen arbitrary[14]. Furthermore it is incremental and deterministic, which means that smaller RDF documents are contained in bigger documents and that the same dataset is created when the generation run is executed with the same parameters.

Representativity. Due to the dictated structure of the dataset it lacks a degree of structuredness real RDF data has, even though the data itself is real RDF data. Furthermore, the queries are close to queries which may be performed on the DBLP.

Dissemination. SP^{2}Bench is freely available for both academic and industrial use.

Intelligibility. The SP^{2}Bench does not have a special way of presenting its results. There is just a list of values for each metric.

Diversity. SP^{2}Bench provides several star shaped queries and complex queries but no purely chain shaped queries. Furthermore, there are metrics for measuring the execution time of queries, the time the dataset needs to be loaded into the store and others which make it possible to identify the differences in different systems.

Criticality. One query of the query set of SP^{2}Bench simply asks for a release year of a certain document, which has traversal length of one. Another query returns all authors that are connected to a given author by at most 3 edges. This query is the query with the highest traversal length and therefore a high traversal length is not given. Thus the criticality criterion is not fulfilled.

3.5 DBpedia SPARQL Benchmark

The DBpedia SPARQL Benchmark (DBPSB) uses the DBpedia knowledge base as dataset and real mined queries to test how well RDF store handle real world queries.

Dataset. The dataset of the DBPSB is based on the DBPedia dataset. DBPedia is a community project of the university of Leipzig, the university of Mannheim, the Hasso-Plattner institute and the open link software team. DBPedia extracts structured information from Wikipedia and makes it available for web applications[15]. The data generator of the DBPSB follows a simple but generic dataset creation process. It starts with an existing input dataset of DBpedia loaded into the official SPARQL endpoint. In order to create datasets of multiple sizes of the input dataset, the existing triples are duplicated and their namespaces are changed. This process can be repeated until the dataset reaches the wished size.

Queries. The queries for the DBPSB are chosen from a query log of all queries posed on the official DBpedia SPARQL endpoint for a three-month period. For the benchmark described in [11] all queries during the period from April to July 2010 are used. In this period 31.5 million queries were posed to the endpoint. Two ways were proposed to obtain a smaller number of queries appropriate for benchmarking triple stores. 1) Queries which are the same or just slightly different are posted to the endpoint. In order to ensure that equal queries are recognized all query variables are renamed in

[14] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/data.php retrieved at 23.01.16
[15] http://wiki.dbpedia.org/about retrieved at 24.01.16

a consecutive sequence as they appear in the query. As a result, distinguishing query constructs such as `DISTINCT` can distinguish more queries. 2) Queries which appear less then 10 times are discarded because they do not contribute much to the overall query performance.
Applying both methods on the query log reduces the number of queries to 35,965. In the next step the queries are put in a graph structure according to their similarity. This graph is then checked for clusters. The different clusters represent groups of queries and can be used to generate prototypical queries for a cluster. Finally 25 queries are chosen from the different clusters representing most commonly SPARQL features to create a query mix. The final queries are freely available[16].

Execution strategy. The execution strategy of the DBPSB is divided into 3 steps. First the triple store under test and all associated programs are restarted in order to clear memory cache. The second step is a warm up phase of 20 minutes, in which queries distinct from the benchmark queries are executed to warm up the store. The third step is the hot run phase in which the chosen queries are executed and metrics are measured. The duration of the hot run phase is 60 minutes.

Metrics. The performance of triple stores is evaluated with respect to two main metrics.

- Overall performance: The overall performance of a triple store which is measured by computing the query mixes per hour. Queries which failed were penalized with 180 seconds.[12]
- Query-based Performance: The metric is used to illustrate the query-based performance and represents the queries per second. If $Qmph$ is the query mix per hour value, q is a query, n is the amount of executed queries and $t(q)$ is the execution time of the query q then the query-based performance is defined as: $\frac{\sum_{i=1}^{n} q_i}{Qmph}$.

Evaluation

The quality criteria listed in section 3.2 are fulfilled to the extend described in the following list:

Goal. DBPSB uses the DBpedia knowledge base as dataset and query-log mining, in order to test how well a store can handle real world queries.

Continuity. The DBPSB was released in 2011 and has no newer version since then. Due to the fact that SPARQL 1.1 was introduced in 2013 the continuity criterion is not fulfilled.

Syntactical correctness. The queries and the dataset are error-free but the queries lack a natural language description. The queries are written in SPARQL 1.0 and the dataset is written in RDF/xml.

Independence. No use case is defined for this benchmark. Furthermore, the dataset is synthetic and it uses queries that were posted to the official SPARQL endpoint of DBpedia, which only allows queries that are easily executable on the DBpedia dataset. Therefore the independence criterion is not fulfilled.

Scalability. As mentioned before the dataset is scaled by replicating and renaming existing data. Hence, the dataset gets bigger but the structure is replicated.

[16] https://svn.code.sf.net/p/akswbenchmark/code/AKSWBenchmark/queries/BenchmarkQueries retrieved at 19.01.16

Representativity. The dataset is based on the DBpedia dataset which even solves the structuredness problem described by Duan et. al. because the data is not necessarily well structured as it is in a purely syntheticly generated dataset [9]. But when the dataset is scaled, a part of the dataset is simply replicated and thus, the representative features of the dataset might get lost. Furthermore, the queries used in the benchmark are extracted from 31.5 million real queries. This makes the queries mimic real world queries and fulfil the representativity criterion.

Dissemination. The benchmark is freely available[17].

Intelligibility. The resulting metrics of the benchmark are not presented in a special way but simply by an output of numbers.

Diversity. The diversity of queries partly depends on the queries posed to the DBpedia SPARQL endpoint. But the query generation process chooses diverse queries from the query log, thus it is assumable that queries with the different query graph structures are generated but it is not guaranteed. Finally the metrics are rather limited due to the fact, that the benchmark focuses on only two main metrics, which both describe the execution time of queries.

Criticality. Due to the fact that the queries are representative and newly generated for benchmark runs nothing can be said about critical queries in general. But due to the fact that the queries posted to the SPARQL endpoint are usually rather short, it is unlikely that queries with a high traversal length are created but not impossible.

3.6 Lehigh University Benchmark

The Lehigh University Benchmark (LUBM) is developed to evaluate the performance of RDF stores with respect to queries over a dataset which represents an ontology[18][19].

Dataset. The LUBM focuses on benchmarking stores for big Web Ontology Data (OWL) data [13]. OWL is a semantic markup language for publishing and sharing ontologies on the world wide web[20]. The ontology used in the benchmark is called Univ-Bench. Univ-Bench describes universities, departments and the activities that occur at them. The dataset is created with a data generator called Univ-Bench Artificial data generator (UBA). UBA creates a graph based on the ontology with the number of universities as scale factor. Furthermore, there are some restrictions to make the dataset more realistic. An example is a minimum of 15 and a maximum of 25 departments at each university.[13]

Queries. There are 14 test queries written in SPARQL. The queries were chosen in respect to

- The size of the queries: This is measured as the proportion of the class instances involved in the query to the total class instances in the benchmark data.
- The selectivity: This is measured as the estimated proportion of the class instances involved in the query that satisfy the query criteria.

[17] http://aksw.org/Projects/DBPSB.html retrieved at 24.01.16

[18] In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. http://web.dfc.unibo.it/buzzetti/IUcorso2007-08/mdidattici/ontology-definition-2007.htm retrieved at 20.02.16

[19] http://swat.cse.lehigh.edu/projects/lubm/ retrieved at 28.01.16

[20] https://www.w3.org/TR/owl-ref/ retrieved 28.01.16

- The complexity: The complexity is illustrated by the number of different classes and properties in a query.
- The assumed hierarchy information: This considers whether information from the class hierarchy or property hierarchy is required to achieve the complete answer.
- The assumed logical inference. This considers whether logical inference is required to achieve the completeness of the answer.

The queries were chosen to cover a wide range of the factors listed above.

Execution strategy. First several datasets of different sizes with 1 university, with 5, 10, 20 and 50 universities are created. LUBM uses a tester for the execution of the benchmark. The tester requests operations on the repository (e.g., open and close), launches the loading process, issues queries and obtains the results [13]. Due to the fact, that some of the stores benchmarked in [13] did not support SPARQL when they were tested, queries had to be translated manually into supported query languages. Then the queries are executed on the different datasets without any mentioned warm up phase and the metrics are measured.

Metrics. The LUBM contains a set of metrics, including a combined metric for query performance.

- Load time: The time needed to load the dataset into the system.
- Repository size: The size of the repository of the system under test after loading the dataset.
- Query response time: The response time for each query of the query set. Each query is executed 10 times and the average execution time is returned.
- Query completeness and soundness: The completeness is the percentage of all returned possible query results. The soundness is the percentage of correct query answers which are actually returned.
- Combined metric: For the combined metric the query response time, the completeness and soundness are combined. The combined metric enables an easier comparison and makes it possible to rank different systems. It is stated that such a combined metric should be used carefully because it leaves out certain aspects and emphasizes different.[13]

Evaluation

The quality criteria stated in section 3.2 are fulfilled for the LUBM to the extend described in the following list:

Goal. LUBM tests how well stores can handle OWL data and how well stores can draw inferences from that data.

Continuity. LUBM was updated the last time in 2005 [13]. Due to the fact that SPARQL became a World Wide Web Consortium Recommendation in 2008[21] and SPARQL 1.1 was introduced in 2013 there is no continuity in this benchmark.

Syntactical correctness. The queries and the data are error-free. The queries are described in SPARQL 1.0 and a natural language description and explanation of the queries can be found in the appendix of [13]. Furthermore, the dataset is written in OWL.

[21] https://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation/ retrieved at 25.02.16

Independence. LUBM is designed around a use case. This use case is dictated by the synthetic dataset. Furthermore, the queries of this benchmark were chosen in respect to several factors. The factors by which the queries were chosen are comprehensible and thus reduce the probability the queries were designed along a given system.

Scalability. The dataset can be scaled by changing the amount of universities in the dataset.

Representativity. The queries and the dataset have ample criteria, which should ensure that the data and queries are realistic. Even though there is an effort to mimic real data, the data is still synthetic and does not cope with the structuredness problem described by Duan et al. [9].

Dissemination. The LUBM is freely available[22].

Intelligibility. Nothing is said about the representation of the resulting data.

Diversity. The 14 test queries of LUBM include queries, which create star shaped, chain shaped and complex query graphs[23][24]. Furthermore, LUBM provides diverse metrics and even an combined metric, which measure several different aspects of the tested system. Thus the diversity criterion is fulfilled.

Criticality. The traversal length of the queries range from very low to higher traversal lengths of 4 with more complex queries. Thus the criticality criterion is fulfilled[25].

3.7 Social Network Intelligence Benchmark

The Social Network Intelligence Benchmark (SIB) takes the schema of popular social networks, e.g. Facebook, to create an RDF dataset for benchmarking[26].

Dataset. The dataset mimes a common social network like Facebook. The name, age, gender and homeland of a user of the social network are randomly created from a dictionary by a data generator. Furthermore, each user specifies 0 to 10 singers he likes. The information about singers are taken from DBpedia to mimic a realistic information base and network. Between different users of the network a friendship relationship can be established. It is also possible for users to create events, groups and upload photos. Each of these possibilities mimics Facebook components[27]. The size of the dataset is scalable by the amount of users in the dataset.

Queries. The benchmark tests ample queries with three query mixes which are expressed in SPARQL 1.1. In addition to `SELECT` queries SIB tests other query forms like `ASK` `DESCRIBE` or `UPDATE` queries.
1) The interactive query mix consists of 20 queries which are used by users in order to find information in the social network. For instance: Find all users whose first names contain a particular string. There are less complex queries like the one just stated or more complex ones, e.g.: Find people having the same gender, having a similar age, living in the same areas, who are not friends of a specific user, and order them by the number of shared interests with the user.
2) The update query mix consists of 10 queries. For instance, it enables the user to

[22] http://projects.semwebcentral.org/projects/lubm/ retrieved at 20.02.16
[23] http://swat.cse.lehigh.edu/projects/lubm/lubm.jpg retrieved at 25.02.16
[24] http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt retrieved at 25.02.16
[25] http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt retrieved at 20.02.16
[26] https://www.w3.org/wiki/Social_Network_Intelligence_BenchMark retrieved at 30.01.16
[27] https://newsroom.fb.com/products/ retrieved at 30.01.16

update his profile information or to create a group and add friends to it. As in the interactive query mix the complexity of the different queries cover a wide range.
3) The analysis query mix consists of 11 queries which retrieve information about products and users. For instance, which users are iPhone users or potential iPhone clients? Or as a more complex query: What is the topic with the most users, who have joined in the last day? A user is considered to have joined if the user has not been discussing the topic in the past 10 days.
All queries are executed multiple time. The amount of times a certain query is executed differs because some queries, like find all users whose first names contain a particular string, are more popular in real networks than other more specific queries.
Finally it is necessary to mention, that even though it is stated in [14] that the queries are written in SPARQL 1.1 no property paths were used in the queries.

Execution strategy. No default benchmark execution strategy is proposed for the SIB.

Metrics. The SIB consist of only three metrics:

- Query per second: How many queries are executed per second.
- Query mix per hour: How many query mixes are executed per hour.
- Total execution time: How long does it take to execute all query mixes and the contained queries as often as their popularity demand it.[14]

Evaluation

The SIB fulfils the quality criteria for benchmarks listed in section 3.2 as described in the following list:

Goal. The SIB tests how well stores can handle different query forms like `SELECT, ASK` and `UPDATE` with queries, which mime a social network.

Continuity. Due to the fact that [14] was updated in 2013 for the last time with version 0.8 and no Property Paths or other features of SPARQL 1.1 are used in the benchmark the continuity criterion is no fulfilled.

Syntactical correctness. The dataset and queries are error-free. The queries are accompanied by a natural language description and written in SPARQL 1.1. The dataset is stored in the n-triple format.[28].

Independence. SIB is designed around use case that mimes a social network like Facebook. The dataset of SIB is designed along several factors to mime a real social network and is scalable. Furthermore, several properties of the dataset are created randomly in each generation process. Nevertheless, the dataset is still synthetic and could be designed in order to support a certain system, but the aspects stated before make it rather unlikely. Similar to the dataset, the queries were designed to mime real queries, which could be executed on a social network and thus are designed along the use case.

Scalability. The social network like dataset is scalable with the amount of users in the network as scale factor.

Representativity. The dataset is very close to a real social network. But nevertheless, it is still completely synthetic data. Thus, the structuredness problem described by Duan is not solved[9]. Furthermore the queries are fixed, but close to the given use case.

Dissemination. The SIBM and its implementation is freely available[29].

[28] https://www.w3.org/wiki/Social_Network_Intelligence_BenchMark
[29] https://sourceforge.net/projects/sibenchmark/ retrieved at 20.02.16

Intelligibility. In [14] nothing is stated about how the resulting data is represented.

Diversity. SIB delivers queries, which create star shaped, chain shaped and complex query graphs. But due to the fact that all three metrics focus on the execution time of queries the SIB is limited in its capability of finding differences in different systems.

Criticality. The queries in each query mix range from rather simple queries to complex queries as described in section 3.7. One query is: Find people who studied at the same school that connect with you by a friend relationship. It is even stated that this query can be expressed with Property Paths to arbitrary alter the length of the friendship relationship path, but the query itself does not use Property Paths. Thus, even though it is stated how to create queries with a high traversal length, there are no such queries by default.

3.8 Waterloo SPARQL Diversity Test Suite

The Waterloo SPARQL Diversity Test Suite (WatDiv) offers a wide range of queries with varying structural characteristics of the dataset. It is available in the newest version 0.6 from 2015[30].

Dataset. The WatDiv data generator allows users to define their own dataset through a dataset description language. This language makes it possible to create a dataset based on individual needs. With this language ample properties of the dataset can be controlled:

- What the scale factor or factors of the dataset are.
- Which types of entities should be part of the dataset and how many of their instances will occur in the dataset.
- The structuredness of each entity.
- How different entities are associated.
- The probability that an entity of type X is associated with an entity of type Y.
- The cardinality of such associations.[31]

An important characteristic that distinguishes the WatDiv test dataset from other benchmarks is that instances of the same entity do not necessarily have the same set of attributes[32].

Queries. The benchmark queries are generated in two steps. In the first step, ample query templates are created by performing a random walk over the graph representation of the benchmark data. In the second step place holders of the templates are filled with RDF data from the dataset [15]. In [15] 12500 test queries from a total of 125 query templates were created for benchmarking.

Execution strategy. The benchmark execution strategy proposed in [15] starts with creating 2 datasets with 10M and 100M triples. The following steps were executed on both datasets. The dataset is loaded into the tested store and a warm up phase is started in which the 12500 created queries are executed. Then the same 12500 queries are executed on the warm store in randomized order and the metrics are measured. This step is repeated 5 times to achieve higher confidence.

[30] http://dsg.uwaterloo.ca/watdiv/ retrieved at 30.01.16
[31] http://dsg.uwaterloo.ca/watdiv/watdiv-schema-tutorial retrieved at 30.01.16
[32] http://dsg.uwaterloo.ca/watdiv/ retrieved at 30.01.16

Metrics. The metrics of the WatDiv are neither on the homepage nor in [15] directly stated. But from the representation of the result data in [15] the following metrics are assumable.

- Per-query performance: The amount of time needed for the execution of single queries.
- Total workload execution time: The total amount of time needed to execute all queries.

Evaluation

In regard to the quality criteria stated in section 3.2 the following can be said.

Goal. WatDiv allows to create a dataset and queries according to the aims of the user.

Continuity. Even though new versions of the DivWat are available regularly and the last update on the homepage of WatDiv was in November 2015 no Property Paths are considered in this benchmark. Thus, there is continuity in this benchmark, but it lacks critical aspects[33].

Syntactical correctness. The language used for creating the dataset is error-free and makes it possible to create error-free dataset. The queries are newly generated for each benchmark process. Thus, there is no available natural language description of the specific queries, but a set of query templates is described[34]. Furthermore, the dataset is stored in the n-triple representation.

Independence. Due to the fact that the user can decide on several criteria like the structuredness and the scale factors of the dataset and that the queries are newly generated for each benchmark process, the user can define his own use case and therefore the independence criterion is fulfilled.

Scalability. Not only that the size of the dataset is scalable, but also the frequency of instances of the different types can be chosen.

Representativity. Even though the dataset of the WatDiv is created synthetic the data generation language gives the possibility to customise the dataset in such a way that it mimics real world data. Even the structuredness problem described by Duan in [9] is solved by giving the possibility to determine the structuredness of each entity in the dataset. The queries are created according to the dataset. Thus representative queries are possible but not guaranteed.

Dissemination. WatDiv is freely available[35].

Intelligibility. Nothing is said about the representation of results neither on the homepage nor in [15].

Diversity. The query templates of WatDiv create queries with star shaped, chain shaped and complex query graphs[36]. Nevertheless, WatDiv lacks metrics, which measure different aspects of the tested system.

Criticality. Due to the fact, that the queries are newly generated for each benchmark process, complex queries are possible but not guaranteed.

[33] fulfilleddsg.uwaterloo.ca/watdiv retrieved at 30.01.16
[34] http://dsg.uwaterloo.ca/watdiv/basic-testing.shtml retrieved at 30.01.16
[35] http://dsg.uwaterloo.ca/watdiv/watdiv_v06.tar retrieved at 20.02.16
[36] http://dsg.uwaterloo.ca/watdiv/basic-testing.shtml retrieved at 25.02.16

3.9 Semantic Publishing Benchmark

The Semantic Publishing Benchmark (SPB) is a benchmark developed by the Linked Data Benchmark Council[37] which was inspired by the media and publishing industry[38]. It was designed along a media or publishing organisation scenario in which news, articles and media assets are updated regularly. These news, articles and media assets are enriched with meta data that describes them.

Dataset. For the data generation two components are needed in the SPB. First, ontologies which define the relations between entities in the reference data. And second the reference datasets, which are collections of entities which describe ample domains, e.g., sports, persons, politics, geo-locations, etc. These components are the bases of the dataset generation process. The generated data is named as creative work and is defined as meta-data about a real entity (or entities) that exist in the reference datasets [16]. The dataset is scalable by the number of entities. Each creative work can have several properties like title or description which are literals. Furthermore, each creative work can have properties like mention or about, which link it to entities from the reference dataset. The data generator models three types of characteristics in data :

- Clustering of data: This effect is achieved when creative work about a single entity is produced. In the beginning a very high amount of creative work is generated, but then the amount of generated creative amount is decreasing. This is done to mime reactions on news which have their peak of interaction when the news are released. People loose interest in news when they are older[16].
- Correlations of entities: This effect is achieved by generating creative work about 2 or 3 entities in a certain amount of time. Each entity is tagged by a creative work in the beginning, the end and in the middle of the time period. This should mime ample artists who are working together on a project in a fixed amount of time.
- Random tagging of entities: The Random tagging focuses on "popular" entities. 5% of all entities are declared as popular and receive 30% of all random tags, whereas the remaining 95% of entities, which are not "popular" receive 70% of all random tags.

In the data generation process itself the reference data is first queried in order to identify the entities in the dataset. Then one of the three data characteristics described above is used to create creative work for every entity.

Queries. 2 sets of queries exist for the SPB. A basic query mix consisting of 9 queries and an advanced query mix consisting of 25 queries. SPB defines so called choke points. Choke points are defined as technical challenges that each database needs to overcome in order to satisfy the need for fast and reliable service[16]. The queries in the 2 query mixes are designed to test if the choke points are problem points in an RDF store. The choke points are:

Join ordering. Tests the ability to consider cardinality constraints and decide which type of join to be used.

Aggregation. Tests the ability to evaluate sub-selects first.

Optionals and nested optionals. Tests the ability of the query optimizer to produce a plan where the execution of the triple patterns inside optional section is the last to be performed since they do not affect the size of the intermediate results.

[37] http://www.ldbcouncil.org/ retrieved at 31.01.16
[38] http://ldbcouncil.org/developer/spb retrieved at 31.01.16

Parallel execution of unions. Tests the ability of optimizer to produce query plans where unions are executed in parallel. This is helpful if the involved sub-queries produce a large number of results.

Ordering. Tests the ability of the optimizer to choose query plan that facilitates the ordering of results.

Geo-spatial predicates. Tests the ability to handle queries for geo-spatial data, mentioning entities within a geo-spatial range.

Full-text search. Tests the ability of the systems to utilize custom optimizations e.g. indexing when dealing with text search.

Duplicate elimination. Tests the ability to identify duplicate entries during the creation of intermediate results.

Complex filter conditions. Tests the ability of query optimizer to split complex filter conditions into conjunction of conditions and execute them in parallel as soon as possible.[16]

The queries are written in SPARQL 1.1, but they contain no Property Paths. Additionally to these queries two agents exist which execute queries during the whole benchmarking process. The queries these agents execute are INSERT, UPDATE and DELETE operations.

Execution strategy. First the dataset is created and loaded into the tested store. During the benchmarking process two agents execute queries on the dataset constantly. One agent is an editorial agent, which executes queries for updating the dataset and deleting or inserting information to it. The queries are distributed to 80% INSERT operations, 10% UPDATE operations, 10% DELETE operations. The other agent is an aggregation agent, which simulates end-users, which look up information. This agent executes queries which look up creative work about certain entities or creative work of a certain date. At the same time the queries of one of the query sets are executed and the metrics are measured.

Metrics. SPB consist of the following metrics:

- Maximum, minimum, and average execution times for each operation and query of the agents during the whole benchmarking process.
- The average number of queries per second and the total amount of queries.
- The number of returned results for each query.

Evaluation

The following list represents to what extent the quality criteria stated in section 3.2 are fulfilled by the SPM.

Goal. SPM defines the in section 3.9 mentioned choke points and tests how well a store can handle those.

Continuity. Since the newest version of the documentation for SPM 2.0 was uploaded in November 2015 and SPARQL 1.1 is used within it, but Property Paths are ignored, the continuity criterion is fulfilled only partially.

Syntactical correctness. The dataset and the queries are error-free. Furthermore, the queries of the query mixes are accompanied by a natural language description[16]. The queries of the agents are not described by a natural language description, since

they are newly constructed for every benchmark process. But a schema which describes how they are constructed is given. The queries are expressed in SPARQL 1.0 and the dataset is stored in the n-triple format.

Independence. The SPM is designed around a media or publishing organisation use case and therefore the synthetic dataset and all queries are designed along this use case.

Scalability. The dataset is scalable and uses the amount of entities as scale factor in the dataset.

Representativity. Even though the dataset and queries are designed close to a real use case, and the queries are executed continuously by the agents, the dataset and the queries are still synthetic. Thus, the structuredness problem described by Duan in [9] is not solved.

Dissemination. The SPM is freely available[39].

Intelligibility. During the whole benchmarking process the newest results are displayed on the console. Furthermore, 3 log files are created:

- A query log file which contains all executed queries, the execution time and the number of returned results for each query.
- A detailed query log which contains all results for all queries.
- A result which contains the total execution time of the benchmark, the average, minimal and maximal execution time per query, the average queries per second and their total amount and if the results of the benchmark execution are correct results.

Diversity. The fixed query mixes contain queries, which create star shaped, chain shaped and complex query graphs. Furthermore the metrics of SPM measure different aspects of the system. Thus the diversity criterion is fulfilled.

Criticality. Due to the fact, that the queries were designed along several choke points, a certain complexity is given, but a high traversal length cannot be found in any query.

3.10 SPARQL Linked Open Data Query Generator

SPARQL Linked Open Data Query Generator (SPLODGE) is a tool set for the systematic generation of SPARQL queries which cover a wide range of possible requests on the Linked Data cloud [17]. Even though SPLODGE itself is not a benchmark, it is a part of a benchmark which can be combined with any other components to create a benchmark.

Dataset. SPLODGE can be used with any dataset.

Queries. In contrast to the queries of other benchmarks which focus on centralized RDF stores, SPLODGE focuses on the generation of queries for a federated RDF store, i.e., a database which consists of ample independent RDF stores and a query federator that queries the different RDF stores separately and combines the returned results. As bases for the query generation several query characteristics are stated which are summarized in 3 groups.

Query algebra. This group relates the semantic properties of SPARQL. It contains the query type, which can differ between SELECT, DESCRIBE, ASK and CONSTRUCT, the join type, which can differ between conjunctive join(.), disjunctive

[39] https://github.com/ldbc/ldbc_spb_bm_2.0 retrieved at 31.01.16

join (UNION), and left-join (OPTIONAL), and the result modifiers DISTINCT, LIMIT, OFFSET, and ORDER_BY.

Query structure. The second set of characteristics describes how basic graph patterns are combined. It consists of the variable patterns, which describe where variables occur in the query, the join patterns, which describe how the same variable is used within different triples to create joins, and the cross products, which describe conjunctive joins over triple patterns which do not share a common variable.

Query cardinality. The last set of characteristics deals with the the number of sources, the number of joins, and the query selectivity. The query selectivity denotes the proportion between the overall number of triples in a graph and the number of matched triples by a query.

These characteristics are used to create common and critical queries.

Metrics. The metrics can be chosen freely.

Execution strategy. Due to the fact that SPLODGE only creates the queries nothing can be said about an execution strategy for benchmarks using SPLODGE.

Evaluation

Due to the fact that SPLODGE itself is no complete benchmark but a component of a benchmark, just some of the quality criteria are addressed.

Goal. SPLODGE generates representative queries especially for federated RDF stores and thus is used for testing federated RDF stores.

Continuity. Due to the fact, that SPLODGE is just a tool for query generation and not a complete benchmark the continuity criterion can be ignored.

Syntactical correctness. The quality of the dataset depends on the chosen dataset, thus there can be nothing said about it. The queries are correct and error-free and lack a natural language description because the queries are newly generated in each process.

Independence. Due to the wide range of queries which are generated with SPLODGE, and the fact that the other components of the benchmark can be chosen freely, the user can define his own use case and therefore the independence criterion is fulfilled.

Scalability. If a benchmark fulfils the scalability criterion depends only on the dataset, thus nothing can be said about the scalability criterion.

Representativity. The query characteristics addressed by SPLODGE make it possible to create common queries for most use cases. Therefore the representativity criterion is fulfilled.

Dissemination. SPLODGE is freely available[40]

Diversity. The different characteristics SPLODGE defines for the creation of queries make it possible to create queries with any query graph. Therefore the diversity criterion is fulfilled in regard to the queries. Due to the fact that no metrics are defined for SPLODGE nothing can be said about the diversity of queries.

[40] https://github.com/goerlitz/splodge retrieved at 9.02.16

Criticality. As described in [17] SPLODGE can be used to create corner case queries, i.e., queries which test the limitations of a system. When the defined query characteristics are chosen rightly, critical queries with a high traversal length can be created.

3.11 Graph Database Benchmarks

Some graph database benchmarks traversal queries exist, which are similar to Property Paths. Thus graph database benchmarks may contain interesting aspects for the creation of a Property Path benchmark.

The high performance computing Scalable Graph Analysis Benchmark (SGAB) executes 4 different operations on a weighted, directed graph. The first operation is the graph creation. The second operation creates a list of the edges with the highest weights. The third operation applies a k-hop algorithm to each edge in the edge list and subgraphs are created, which consist of the vertices and edges of all the paths of length k. The final operation creates a list of nodes with the highest betweenness centrality score. The betweenness centrality score determines how central a vertex is in a graph. The duration of each operation is measured and during the fourth operation the traversed edges per second are measured [18]. Due to the fact that this benchmark focuses on the weights of the edges of a graph and that edges in RDF do not have weights, this benchmark is not useful for the creation of a SPARQL Property Path benchmark.

The Graph 500 benchmark focuses on weighted undirected graphs. Similar to SGAB it executes different operations on a graph and measures some metrics. In the first step of the benchmark an edge list is randomly created from which a graph is created and loaded into the store of the tested system. The time this process takes is measured. After that the benchmark creates 64 random search keys. For each key all parents are computed and a parent array is created. The time this process takes is measured. Finally, the parent array is validated and performance information are returned[41]. This benchmark is not useful for the development of a benchmark for Property Path because the edges of the graph are undirected and weighted.

Marek Ciglan's Graph Traversal Benchmark for Graph Databases claims that the goal of the project is to provide a benchmarking mechanism to measure effectiveness of graph databases for graph traversal operations[42]. The dataset of the benchmark is a network with realistic properties, which is created by a data generator. Then traversal operations are executed on the graph. It focuses on two operations: A breadth-first traversal starting at a single vertex and a whole graph traversal. The time these operations take and different metrics are measured [19]. Even though the benchmark contains traversal operations, it is of limited use for the creation of a new benchmark because it only has two query like structures and the fact that the edges and vertices of the graph contain properties.

Ample graph database benchmarks turned out as unprofitable for the creation of a benchmark for SPARQL property paths due to different reasons.

[41] http://www.graph500.org/specifications#sec-1 retrieved at 11.02.16
[42] http://ups.savba.sk/ñarek/gbench.html

3.12 Summary

In order to summarize the evaluated benchmarks in regard to the quality criteria mentioned in section 3.2 the following can be stated:

Goal. Each benchmark has a goal and WatDiv even makes it possible for the user to define his own goals.

Continuity. For most benchmarks the continuity criterion is not fulfilled. Only WatDiv and SPB are updated continuously. But even though they are updated regularly they do not consider all aspects of SPARQL 1.1 , e.g., Property Paths are ignored by both benchmarks.

Syntactical correctness. BSBM, SP^{2}Bench, LUBM, SIB, WatDiv and SPB are syntactically correct, which means that they fulfil this quality criterion to full extent. The DBPSB and SPLODGE lack a natural language description of the queries but fulfil all the other aspects of the criterion as well as the other benchmarks do. It has to be added that SPLODGE generates new queries in each benchmark process and therefore cannot provide a general natural language description for each query.

Independence. The independence criterion is fulfilled, when the dataset and queries of a benchmark are not influenced by an existing RDF store. SPLODGE and WatDiv are independent, due to the fact that the user can specify the dataset and the queries of the benchmarks depending on several criteria. Due to the fact that DBPSB is the only benchmark that uses a real world dataset, the dataset can be seen as independent. Synthetic datasets are more likely influenced by existing RDF stores. In regard to the queries SP^{2}Bench uses queries, which especially test technological challenges of RDF stores. Due to the fact that these challenges are independent from an implementation in particular, the queries can be seen as independent. The queries of BSBM , LUBM, SIB, SPB are chosen in regard to comprehensible use cases. But the independence of those queries cannot be assured. DBPSB generates the queries from a query log. Due to the fact that this log contains queries, which can be executed efficiently by the underlying system, these queries are depending on this system.

Scalability. Each benchmark has a scalable dataset. BSBM, LUBM, SIB and SPB have a scale factor, which means that the amount of a certain type of vertex is altered. In the case of BSBM it is the amount of products in the dataset and in LUBM it is the amount of universities. SP^{2}Bench creates the dataset from an existing dataset and simply copies the needed size or triples into the tested store. Thus, the same dataset is created in each run with the same parameters. DBPSM uses a real dataset too. But in the case of DBPSBM a part of the DBPedia dataset is chosen as bases for the dataset creation. For scaling the dataset this portion is then replicated until the dataset of the desired size is created. Finally, WatDiv provides a dataset description language. This language makes it possible to alter frequency of instances of different types.

Representativity. The dataset and the queries are the decisive aspects for evaluation of the representativity criterion. A dataset can be simple like the dataset of the BSBM, which means that it is synthetic. Furthermore a dataset can be complex like the dataset of LUBM, SIB, WatDiv, SP^{2}Bench and SPB. This means that the dataset is synthetic as it is in a simple dataset, but has to fulfil several criteria to mime a real world dataset. Finally a dataset can be real RDF data as it is in DBPSB. Queries can be close to a use case like they are in BSBM, SIB, and SPB, which means that they were designed along the use case. Furthermore, SP^{2}Bench, LUBM and SPLODGE create or choose their queries in respect to several characteristics. Several of these characteristics were

chosen to create representative queries. DBPSB creates the queries from a query log of real queries that were posted to a SPARQL endpoint. WatDiv uses query templates to create queries.

Dissemination. Due to the fact that each benchmark is available freely, the dissemination criterion is fulfilled for each benchmark.

Intelligibility. The intelligibility criterion is fulfilled by BSBM and SPB. Both benchmarks describe the way the measured metrics are represented. All other benchmarks simply return a list of values as result or nothing is stated about the format of the results at all.

Diversity. The diversity of each benchmark depends on the diversity of queries and the diversity of metrics. Queries of a benchmark should create all three query graph structures in order to provide diverse queries. LUBM, SIB , and SPB provide queries, which create all three query graph structures. Furthermore, WatDiv does not provide queries in particular, but query templates, from which queries are created. These templates are designed to cover all query graph structures. BSBM and SP^{2}Bench provide queries that create star shaped and complex query graphs but lack queries that create chain shaped query graphs. The diversity of DBPSB depends on the queries that were posted to the SPARQL endpoint and therefore a diversity of queries is not guaranteed. Nevertheless the query generation process describes a way that chooses diverse queries from a very high amount of real posted queries and therefore it is assumable that queries, which create diverse query graphs, exist. Finally SPLODGE defines characteristics, which make the creation of diverse queries possible. In regard to metrics DBPSB, SIB and WatDiv only define metrics, which measure a single aspect of the tested system, whereas BSBM, SP^{2}Bench LUBM and SPB define metrics, which measure at least two different aspects of a tested system.

Criticality. The criticality criterion, which is defined by the traversal length of queries, is fulfilled by LUBM. LUBM is the only benchmark containing queries with a traversal length of at least 4, whereas BSBM, SP^{2}Bench, SIB and SPB lack such queries. DBPSB, WatDiv and SPLODGE provide a query generation process, which makes it possible to create queries with a high traversal length. Even though it is possible to create queries with a high traversal length with DBPSB, the queries that are posted to the SPARQL endpoint of DBPedia are usually rather short, thus queries with a high traversal length are unlikely.

In table 1 the result of the study of the different benchmarks is summarized. Due to the fact, that SPLODGE focuses on the creation of queries, some criteria cannot be evaluated for it. These criteria are denoted by -.

Criteria	BSBM	SP^{2}Bench	DBPSB
Goal	Designed around e-commerce to evaluate realistic queries	Measures how well RDF stores handle real RDF data	Tests how well stores can handle real queries
Continuity	Not given (2011)	Not given (2009)	Not given (2011)
Syntactical correctness	Syntactically correct	Syntactically correct	Lacks natural language description
Independence	Dataset is synthetic Queries in accordance to use case	Dataset is synthetic Queries test technological challenges	Dataset is real world RDF data Queries from query log
Scalability	Scalable by one specific RDF type	Scaled by copying and renaming triples	Renamed triple replications
Representativity	Simple dataset Queries close to use case	Complex dataset Queries based on characteristics	Real dataset Queries from query log
Dissemination	Freely available	Freely available	Freely available
Intelligibility	Output format described	Undescribed list of values	Undescribed list of values
Diversity	Star and complex queries Diverse metrics	Star and complex queries Diverse metrics	Queries depend on SPARQL endpoint Simple metrics
Criticality	No critical queries	No critical queries	Critical queries possible but unlikely

Criteria	LUBM	SIB	WatDiv
Goal	Tests how well OWL data is handled and to what extend reasoning can be done	Tests different query forms	Makes it possible to define own goals
Continuity	Not given (2005)	Not given (2013)	Given but ignores property paths (2015)
Syntactical correctness	Syntactically correct	Syntactically correct	Syntactically correct
Independence	Dataset is synthetic Queries in accordance to use case	Dataset is synthetic Queries in accordance to use case	User can specify dataset and queries based on several criteria
Scalability	Scalable by one specific RDF type	Scalable by one specific RDF type	User definable scalability criteria
Representativity	Complex dataset Queries based on characteristics	Complex dataset Queries close to use case	Complex dataset Queries created from templates
Dissemination	Freely available	Freely available	Freely available
Intelligibility	Output format not described	Output format not described	Output format not described
Diversity	Star, complex and chain queries Diverse metrics	Star, complex and chain queries Simple metrics	Star, complex and chain queries possible Simple metrics
Criticality	Critical queries exist	No critical queries	Critical queries possible

Criteria	SPB	SPLODGE
Goal	Test how well several defined choke points are handled	Generates representative queries for federated RDF stores
Continuity	Given but ignores Property Paths (2015)	-
Syntactical correctness	Syntactical correctly	Lacks natural language description (Queries are newly generated)
Independence	Dataset is synthetic Queries in accordance to use case	User can specify dataset and queries based on several criteria
Scalability	Scalable by one specific RDF type	-
Representativity	Complex dataset Queries close to use case	- Queries based on characteristics
Dissemination	Freely available	Freely available
Intelligibility	Output format described	-
Diversity	Star, complex and chain queries Diverse metrics	Star, complex and chain queries possible -
Criticality	No critical queries	Critical queries possible

Table 1: Summary of benchmarks in respect to quality criteria

4 Property Path Benchmark

After several existing RDF stores have been evaluated in regard to the prior introduced quality criteria the new benchmark for SPARQL Property Paths is presented in this section. First, it is discussed how the different elements of the new benchmark have to be be chosen and designed based on the analysis of existing benchmarks. Then the new benchmark is evaluated based on the quality criteria.

4.1 Discussion of Elements of the New Benchmark

In [20] Gray mentions three questions a benchmark should be designed around:

1. What do you want to learn?
2. How much are you prepared to invest?
3. What are you prepared to give up?

The most interesting question at this point is the first one and its answer is: The new benchmark should focus on the capability of a system to handle SPARQL 1.1 Property Paths. Furthermore, the quality criteria defined in section 3.2 should be fulfilled. The answer to the second question is: The benchmark should be developed in the context of this bachelor thesis. And finally the third question: It is stated in [6] and [7] that some quality criteria might be in competition and therefore not all quality criteria might be fulfilled to full extent. In this case the representativity and the scalability of the benchmark are in competition. The focus for this benchmark is the representativity. In order to fulfil the representativity to full extent, the dataset must be real, but a real dataset cannot be scaled arbitrarily. Furthermore, the representativity of the queries could be neglected in order to properly test Property Paths. Due to the fact that Property Paths are not frequently used it is hard to create representative queries that still test each feature of Property Paths. Thus, it can be said that the scalability and the representativity of the queries are the elements that can be neglected.

4.2 Dataset

The dataset of the new benchmark should be syntactically correct, scalable, representative and independent from any RDF store in order to fulfil the quality criteria. Due to the fact that each of the presented datasets is syntactically correct this aspect can be neglected for the discussion. As mentioned in section 4.1 the scalability and representativity of the dataset could not be fulfilled at the same time and therefore compromises have to be done. Due to the fact that the purely synthetic datasets do not solve the structuredness problem defined by Duan in [9] the datasets most interesting for the new benchmark are the datasets of DBPSB and WatDiv. The data creation process of WatDiv allows to create a dataset in accordance to ample criteria, which can be chosen by the user. Nevertheless the dataset is still synthetic and thus, does not fulfil the representativity criterion. Finally the dataset of DBPSB is real RDF data. But when the dataset is scaled it is simply replicated and renamed. In this process the structure is also replicated and thus the diverse structure of real world RDF data might get lost. Therefore, there is no representative dataset with a scaling process that keeps the representativity of the dataset in the presented benchmarks.

In order to have a dataset that fulfils the criteria to a desired degree, the Billion Triple Challenge (BTC) dataset will be used for the benchmark [21]. The dataset was crawled during February to June 2014. The data is stored in the n-quad format. The n-quad format is similar to n-triples but with the difference that it has an additional term, which indicates to which graph the triple belongs[43]. From a set of seed IRIs the dataset was crawled in 14 iterations. In the first iteration the seed IRIs were used as subjects of triples and all resulting triples were added to the dataset. In the next iteration the former crawled objects of the triples were used as subjects and new triples were crawled. Additionally to the triples a term which denotes to which graph the triple belongs is stored. These iterations are also referred to as hops. In these 14 hops a dataset of about 4.000.000.000 quads was crawled [22]. The dataset is not completely syntactically correct and thus, contains incorrect quads. In order to still provide a syntactically correct dataset for the benchmark a corrected dataset will be provided. The corrected dataset was created by iterating over all quads of the original dataset and deleting all syntactically wrong quads.

Due to the fact that the data is real RDF data the structuredness problem defined by Duan in [9] is solved. Furthermore a scalability can be achieved by altering the number of hops of the dataset. This means that a dataset with the scale factor of 7 hops will contain all data that was crawled until hop number 7. This strategy makes it possible to scale the dataset to predefined sizes and keep the representativity of the dataset, because the dataset will still be purely real world RDF data. Hereinafter, the dataset consisting of the data crawled in the first three hops will be referred to as base dataset. The base dataset will be used to verify the soundness and completeness of the query results.

4.3 Queries

The queries of the new benchmark should be syntactically correct, independent, representative, diverse and critical. Due to the fact that no analysed benchmark uses Property Paths in its queries, new queries have to be defined. Therefore several query characteristics will be defined as it was done in SP^{2}Bench, LUBM and SPLODGE. These characteristics are:

- The queries should test each feature of Property Paths as defined in section 2.2.
- The queries should be diverse in their structure as described in section 3.2.
- The queries should contain critical queries with a traversal length of at least 4.
- In [23] it is stated that nested transitive closure operators can lead to very big intermediate results, which cannot be handled. Thus these queries are ignored.

The new benchmark will have two query mixes. One for the base dataset with fixed queries, which are always the same for each benchmark execution and a second query mix for a differently scaled dataset, which will be generated newly by a query generator.

Fixed Query Mix

The fixed query mix for the base dataset will be accompanied by a natural language description of the queries. Furthermore, the correct results of the queries are needed. Due

[43] https://www.w3.org/TR/n-quads/ retrieved at 4.03.16

to the fact the base dataset consists of over 400.000.000 quads it is not possible to obtain the correct results manually. Hence the queries will be executed on several different RDF stores, which will have stored the base dataset, and if a majority of the RDF stores returns the same results, these results will be assumed to be correct. If such a majority cannot be reached, then nothing can be said about the correct results of the queries.

The following listings contain query templates. In these listings the capitals `S` and `P` followed by a number denote subjects and predicates that later will be filled with IRIs. The variable `?o1` will remain a variable in the actual queries later on. These templates will be used to create the queries for the query mixes:

Query 1 tests the sequence operator of Property Paths with two properties. Furthermore it can be represented as chain shaped query graph and has a traversal length of 2.

```
1 SELECT ?o1 WHERE{
2          S1 P1/P2 ?o1
3          }
```

Listing 8: Query 1

Query 2 tests the alternative operator of Property Paths with two alternatives, `P1` and `P2`. It can be represented as star shaped query graph if the `P1` and `P2` are existing. It has a traversal length of 1.

```
1 SELECT ?o1 WHERE{
2          S1 P1|P2 ?o1
3          }
```

Listing 9: Query 2

Query 3 can be represented as complex query graph and has a traversal length of 2. At this point queries with each of the three query graph structures have been tested.

```
1 SELECT ?o1 WHERE{
2          S1 (P1|P2)/P3 ?o1
3          }
```

Listing 10: Query 3

Query 4 returns all vertices that can be reached by a path consisting of at least one P1 and finally exactly one P2. It can be represented as chain shaped graph and has a traversal length of at least 2.

```
1 SELECT ?o1 WHERE{
2          S1 P1+/P2 ?o1
3          }
```

Listing 11: Query 4

Query 5 tests the ability of a store to handle the ^operator and correctly traverse the inverse path.

```
1 SELECT ?o1 WHERE{
2           ?o1 ^P1 S1
3           }
```

Listing 12: Query 5

Query 6 returns all vertices that can be reached from `S1` by any number of `P1` edges including zero edges and thus, `S1` itself. It creates a chain shaped query graph and can have a traversal length of 0 if no edge labelled with `P1` is existing.

```
1 SELECT ?o1 WHERE{
2           S1 P1* ?o1
3           }
```

Listing 13: Query 6

Query 7 returns all vertices that can be reached from A by any number of either `P1` edges, `P2` edges or `P3` edges. It can be represented as complex query graph.

```
1 SELECT ?o1 WHERE{
2           S1 P1*|P2*|P3* ?o1
3           }
```

Listing 14: Query 7

Query 8 first looks up all vertices that can be reached by at least one edge, labelled with `P1`. From those vertices it looks up all vertices that can be reached by any number of edges labelled with `P2` and returns them. It has a traversal length of at least 1 and can be represented as complex query graph.

```
1 SELECT ?o1 WHERE{
2           S1 P1+/P2* ?o1
3           }
```

Listing 15: Query 8

In table 2 the queries, the features of Property Paths in the queries, the query structure and the traversal length infimum and supremum are summarized. Each feature of SPARQL Property Paths mentioned in section 2.12 appears in the table and therefore, is tested by the queries. Furthermore, the query mix contains queries that can be represented as each query graph structure.

Query	Property path feature	Query structure	Traversal length infimum	Traversal length supremum
1	Sequence	Chain	2	2
2	Alternative	Star	1	1
3	Sequence, alternative	Complex	2	2
4	Sequence, +	Chain	2	d
5	Inverse	Chain	1	1
6	*	Chain	0	d
7	Alternative, *	Complex	0	d
8	Sequence, +, *	Complex	1	d

Table 2: Summary of query templates for the query mix

Where d is an upper bound for the traversal length of a graph.

Query Mix Generated by Query Generator

The queries that will be generated by the query generator will use the same templates. The templates will be filled with constants from the BTC dataset, which assure that queries with a diameter of at least 4 are generated in order to fulfil the criticality criterion. For instance, in order to find constants for `S1` and `P1` in query 6 such that a path with length 4 has to be traversed during query processing, the query in listing 16 is executed first.

```
1 SELECT ?s ?p1 WHERE{
2          ?s  ?p1 ?o1.
3          ?o1 ?p1 ?o2.
4          ?o2 ?p1 ?o3.
5          ?o3 ?p1 ?o4.
6          }
```

Listing 16: A query that ensures that a path of `?p1` edges with the length of at least 4 exists

The variable `?s` is the subject of a triple that has an edge labelled with `?p1`. The object of this triple is then used as subject of a new triple with the same predicate as denoted in line 3. This is repeated until a path of 4 `?p1` edges is traversed. Then the variables `?s` and `?p1` are returned. Executing this query ensures that a path of `?p1` edges with length 4 exists. The results can be inserted in query template 6. Thus, it is ensured that a query with the diameter of at least 4 exists.

Furthermore the query in listing 17 was executed in order to ensure that an inverse path for query 5 exists. Then the results of the query were used to create a new query with the ^operator which guarantees a non empty result set.

```
1 SELECT * WHERE {
2          ?s ?p ?o.
3          } LIMIT 1
```

Listing 17: Query that retrieves a random triple to ensure that the inverse path exists.

4.4 Execution Strategy

First, the base dataset will be loaded into the tested system. The base dataset is needed to have verifiable results for the completeness and soundness. In order to measure the query execution time of the benchmark on a warm store the fixed query mix will be executed twice before the actual metrics are measured. The queries of the query mix will be executed one after another and not in parallel. The query mix will be executed 10 times. This means that query 1 to query 8 are all executed once, before query 1 is executed again. The highest and lowest execution time will be deleted to prevent the effect of outliers. Then the dataset will be scaled to the size of 5 hops. After that, the query generator will generate queries for the new dataset and they will be executed in the same way the fixed query mix will be executed. Then the same procedure will be performed for a dataset of 7 hops in order to test the scalability of the tested RDF store. Finally the resulting metrics will be returned in a human readable log file.

4.5 Metrics

The metrics of the new benchmark should be diverse in order to give an insight in the tested system, but focus on the aspects for Property Paths. Therefore several metrics are defined:

The query soundness. The percentage of right query results that are returned by each query. If C_q is the set of correct results for a query q and R_q is the set of returned results of an executed query q then the query soundness is defined as $s(q) = \frac{|C_q \cap R_q|}{|C_q|}$

The overall query soundness. The arithmetic mean of the query soundness. If q_i is query number i then the overall query soundness is defined as: $os(q) = \frac{\sum_{i=1}^{n} s(q_i)}{n}$

The query completeness. The percentage of all possible query results of the query. If C_q is the set of correct results for a query q and R_q is the set of returned results of an executed query q then the query completeness $c(q) = \frac{|C_q \cap R_q|}{|R_q|}$

The overall query completeness. The arithmetic mean of the query completeness. The overall query completeness is defined as $oc(q) = \frac{\sum_{i=1}^{n} c(q_i)}{n}$

The average execution time per query. The arithmetic mean $a(q)$ of the execution time $t(q)$ of each query q. The average execution time per query will be: $a(q) = \frac{\sum_{i=1}^{n} t(q_i)}{n}$ where i is the ith execution of the query. Queries are aborted after 1 hour and counted as failed queries.

The minimum and maximum execution time of each query. Of all query execution iterations the minimum and maximum execution time will be returned.

Due to the fact that the query completeness and query soundness need verifiable query results and verifiable results are only existing for the base dataset, these metrics will only be measured on the base dataset.

4.6 Evaluation

After the elements of the benchmark were presented the benchmark will now be evaluated in respect to the quality criteria described in section 3.2.

Goal. The goal of the Property Path Benchmark is to test how well RDF store can handle queries containing SPARQL 1.1 Property Paths.

Continuity. The continuity criterion can be neglected due to the fact that it has to be fulfilled over time.

Syntactical correctness. The dataset is stored in the n-quads format and error-free due to the fact that incorrect quads were deleted from the dataset. Furthermore, the queries are written in SPARQL 1.1, error-free, and accompanied by a natural language description. Thus, the syntactical correctness is given.

Independence. The dataset of the Property Path Benchmark is based on the BTC dataset, which consists of real world RDF data. Thus the dataset is independent from any particular RDF store. The queries were designed in order to systematically test support of Property Paths, thus the queries are independent as well.

Scalability. The dataset is scalable with the amount of hops in the BTC dataset. Even though this scalability process does not make it possible to scale the dataset to an arbitrary size, the dataset is scalable and keeps its representativity.

Representativity. The dataset of the Property Path Benchmark consists of real world RDF data and thus is representative. The queries were designed to test the capability of RDF stores to handle Property Paths and do not reflect real world queries. Thus the queries are not representative.

Dissemination. The benchmark will be freely available.

Intelligibility. The metrics of the benchmark are returned in a readable log file.

Diversity. The queries of the new benchmark have each of the three query graph structures. Furthermore, the metrics focus on several different aspects of an RDF store like the query soundness, completeness and the query execution time. Thus the diversity criterion is fulfilled.

Criticality. It can be assured that the queries generated by the query generator have a traversal length of at least 4 by following the strategy described in listing 16 in section 4.3.

In summary it can be said that the new benchmark has a clear goal and fulfils the syntactical correctness, independence, dissemination, intelligibility, diversity and criticality criterion to full extent. It is possible to scale the dataset to several predetermined sizes but not to an arbitrary size. Finally the dataset of the benchmark is representative even when it is scaled, but the queries are designed to test the capability of RDF stores to handle Property Paths and thus are not representative. In table 3 the quality criteria in regard to the Property Path Benchmark are summarized.

Criteria	Property Path Benchmark
Goal	Tests how well RDF stores handle Property Paths.
Continuity	Neglected
Syntactical correctness	Syntactically correct by preprocessing
Independence	Real world RDF data Queries designed to test support of Property Paths
Scalability	Scalable to predefined sizes with hops as scale factor
Representativity	Real dataset Queries based on characteristics
Dissemination	Freely available
Intelligibility	Output format described
Diversity	Star, complex and chain queries Diverse metrics
Criticality	Critical queries exist

Table 3: Quality criteria of the Property Path Benchmark

5 Benchmark Results

After the Property Path Benchmark has been introduced, the results of the benchmark for different RDF stores are presented in the following section. In order to point out how well the different stores support Property Paths the limitations of the different stores are explained first. After that the results and execution times of the queries are evaluated and compared. Finally the scalability of RDF stores is presented in the last part.

As stated in section 3.1 several RDF stores that support Property Paths exist. The Property Path Benchmark has been used to benchmark 4 RDF stores in order to test how well they support Property Paths: Virtuoso 7.2.2 community edition, Apache Jena 3.0.1, Allegrograph 6.0.2 free edition and RDF4J 2.0M1 formerly known as Sesame. The benchmark has been executed on a virtual machine with 8 GB RAM, 500 GB disk space, 4 2.9 Mhz processor cores and Ubuntu 14.04 running on it. The java version on the machine has been 1.8.0.77.

Allegrograph is not suitable for benchmarking with the BTC dataset due to the fact that the free edition of Allegrograph has a limit of 5.000.000 triples and the BTC dataset consists of more triples[44]. In order to still get an insight into Allegrograph a Polish Dbpedia dump that consists of around 1.3 million triples was used to benchmark the store[45]. The DBpedia dump can be found on the appended CD. In order to compare the results to another store, RDF4J was also benchmarked with this dataset additionally to the BTC dataset. Due to the insufficient documentation of RDFox[46], it has not been possible to get the store running in reasonable time and thus it has not been benchmarked at all.

[44]http://franz.com/agraph/allegrograph/ag_commercial_edition.lhtml retrieved at 16.05.16
[45]http://wiki.dbpedia.org/Downloads2015-04 retrieved at 20.06.16
[46]http://www.cs.ox.ac.uk/isg/tools/RDFox/ retrieved at 16.05.16

5.1 Limitations of RDF Stores

During the execution of the benchmark several problems have occurred. The BTC dataset has been too big for the virtual machine to be loaded into different stores in the scalings stated in section 4.4. For instance when the data has been loaded into Virtuoso the store loaded the data very slowly and finally aborted. Furthermore, the loading of the data into the stores has taken much more time than assumed. Apache Jena for instance, has needed 11 days for the base dataset of 460 million triples and presumably would have needed around 54 days for the 2.295 billion triple of the biggest planed scaling. Thus, the scaling of the dataset has been omitted for those stores and they have only been benchmarked with the base dataset. Due to the fact that RDF4j was the only store that could handle larger datasets, it has also been benchmarked with differently scaled datasets.

During the benchmark process of Virtuoso, RDF4j and Allegrograph no errors or exceptions have occurred. During the benchmark process of Jena a `java.lang.OutOfMemoryError: Java heap space` has been thrown whenever a query with the * operator was used. Even after the maximum Java heap space has been increased to 6 GB with additional 8 GB swap space for the virtual machine the error still occurred. In order to see if the query returns any results, the amount of results the query should return has been limited to 100. The results that have been returned by a query of the form `SELECT ?o WHERE {A B* ?o.} LIMIT 100` where A and B are valid IRIs, consisted of 100 times A. Due to this fact it is presumable that the query containing the * operator returns A recursively until the main memory was full. To ensure that this is directly due to the * operator and not an additional cycle in the dataset a query of the same form but with an IRI for B that did not exist in the store was executed. This query still returned 100 times A. This shows, that the * operator is not implemented correctly.

Another limitation is identified by query 5, which contains the inverse path operator ^. It has returned an empty result set for Jena and Virtuoso, even though it was ensured that it was not empty as stated in section 4.3. Thus, RDF4j and Allegrograph were the only stores that could handle the inverse operator. Furthermore, query 4 returned an empty result set for RDF4j when it was benchmarked wit the BTC dataset but a non empty result set when it was benchmarked with the DBpedia dump. As for query 5, it was ensured that the result set for query 4 was not empty for the BTC dataset. Thus, RDF4j failed to execute query 4, which tests the sequence and + operator, on the BTC dataset correctly.

While benchmarking Jena, query 2, 3, 4 and 6 returned no results after 1 hour and thus, were aborted. Thus, there is only a valid result for query 1. Due to the lack of comparable results, Jena will be omitted in the comparison of triple stores.

5.2 Evaluation of Results

In this section the returned results of the different stores are compared to each other and a reference result set.

The result sets of Virtuoso and RDF4j on the BTC dataset are different for almost all queries. For instance query 4 and 5 deliver completely different result sets, because for query 4 RDF4j returns 0 results and for query 5 Virtuoso returns 0 results. For other queries, the

result sets of Virtuoso and RDF4j look similar, but there are several IRIs uniquely in each result set except for query 3 where the result sets are equal.

In order to still have an overview over the soundness and completeness of the query results, a reference result set has been created by executing the queries containing Property Path expressions in respective basic graph pattern form. The reference result sets for queries that contain the * and + operators that traverse an arbitrary amount of edges have been created in a range up to 8 edges.

When P+ is evaluated edges of the type P are traversed sequentially. First all vertices that are connected to the starting vertex by an edge of the type P are added to the result set, then all vertices that are connected to the starting vertex by a path containing two occurrences of P, then 3 etc.. If no further edges of the type P exist, then no further results are added to the result set and the result set is enclosed as defined by the semantics of Property Paths. The same is the case for P*, except that also the starting vertex is returned. By iterating over the traversed edges starting from the starting vertex with BGPs, it was made sure that the reference dataset is complete. This was done for Virtuoso and RDF4j and the results for both stores were equal, such that a reference dataset could be formed. In respect to this reference dataset the soundness and completeness of each store for each query is shown in table 4.

Query	**Virtuoso**			**RDF4j**			**Reference**
	|Results|	Sound.	Compl.	|Results|	Sound.	Compl.	|Results|
1 = P1/P2	2	0.5	0.5	2	1	1	2
2 = P1| P2	4	1	0.66	6	1	1	6
3 = (P1| P2)/P3	3	1	1	3	1	1	3
4 = P1+/P2	6	1	1	0	-	0	6
5 = ^P1	0	-	0	4111	1	1	4111
6 = P1*	6	1	1	1	1	0.166	6
7 = P1*| P2*| P3*	8	1	0.57	14	1	1	14
8 = P1+/P2*	5	1	0.71	7	1	1	7

Table 4: The soundness and completeness for Virtuoso and RDF4j

In table 4 the soundness and completeness of the results of RDF4j is 100% for query 1, 2, 3, 5, 7 and 8. For query 4 RDF4j returned an empty result set and for query 6 it only returns 1 of 6 right results. Thus, the soundness for the results is 100% for query 6 but the completeness is only 16,6%. On the other hand virtuoso has incomplete results for each query but query 3 and 6. As mentioned before query 5 returns no result for Virtuoso, because the inverse operator is not working correctly. For query 1 Virtuoso misses 1 result and even returns a wrong result. For query 2 virtuoso returns 2 correct results but misses a third correct result. For query 7 and 8 Virtuoso returns correct, but incomplete result sets. Thus, it can be concluded that RDF4j evaluates the Property Path expressions for sequence, alternative and inverse paths correctly, whereas virtuoso has problems with returning all right results and in case of query 1 even correct results at all.

In order to also have an insight in the soundness an completeness of the results of Allegrograph and RDF4j on the Polish DBpedia dump, a reference result set was created the same way it was done for Virtuoso and RDF4j on the BTC dataset. In table 5 the amount of correct results for Allegrograph and RDF4j, the soundness and completeness of each store are displayed.

Query	**Allegrograph**			**RDF4j**			**Reference**
	| Results|	Sound.	Compl.	| Results|	Sound.	Compl.	| Results|
1 = P1/P2	1	1	1	1	1	1	1
2 = P1| P2	2	1	1	2	1	1	2
3 = (P1| P2)/P3	2	1	1	2	1	1	2
4 = P1+/P2	1	1	1	1	1	1	1
5 = ^P1	2	1	1	2	1	1	2
6 = P1*	5	1	1	5	1	1	5
7 = P1*| P2*| P3*	6	1	1	6	1	1	6
8 = P1+/P2*	4	1	1	4	1	1	4

Table 5: The soundness and completeness for Allegrograph and RDF4j

The table shows that the result sets of Allegrograph and RDF4j are the same for each store and equal to the reference dataset. Thus, it can be concluded that the soundness and completeness of results is 100% for both stores.

5.3 Evaluation of Execution time

The execution times of Virtuoso and RDF4j can be compared only partly. Due to the fact that Virtuoso fails to return complete and sound results for most queries and RDF4j also fails to do so for some queries, the mere comparison of execution times is not very meaningful. Only for query 3, where the result sets are complete and sound for Virtuoso and RDF4j, it can be said that RDF4j handled the query better with 79ms than Virtuoso with 1122ms. Nothing can be said about the execution times for the other queries. Due to that lack of comparability of the two stores, the execution times are omitted.

Query	**Allegrograph**	**RDF4j**
1 = P1/P2	80	40
2 = P1| P2	68	55
3 = (P1| P2)/P3	87	104
4 = P1+/P2	120	68
5 = ^P1	111	56
6 = P1*	233	75
7 = P1*| P2*| P3*	151	102
8 = P1+/P2*	103	78

Table 6: The average execution time per query for Allegrograph and RDF4j.

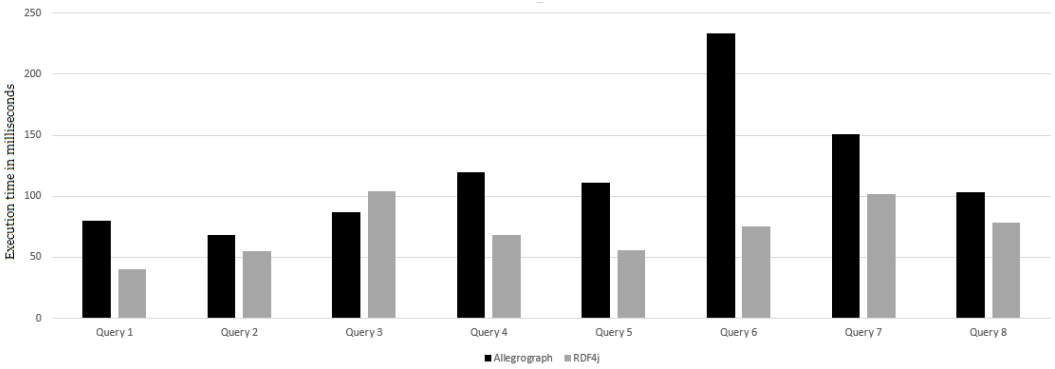

Figure 6: The average execution time for each query for Allegrograph and RDF4j

In table 6 the execution times for each query for Allegrograph and RDF4j with the small DBpedia dump are listed. As can be seen in the table and figure 6 the execution time for Allegrograph is higher for each query but query 3. Due to the fact that both stores return sound and complete result sets, it can be concluded that RDF4j evaluates queries with property path expressions more efficiently than Allegrograph. An interesting point is that RDF4j returns right results on the smaller dataset for query 4 whereas no results were returned by RDF4j when it was benchmarked with the BTC dataset.

5.4 Scalability

Due to the fact that Virtuoso and Jena were not able to handle bigger amount of data than the base data set, RDF4j was the only triple store suitable for benchmarking with a scaled dataset. The dataset was scaled to 5 hops of the BTC dataset, which are about 962 million triple, and then to 7 hops, which are about 1525 million triple. Even though it was described that for each scaling new queries will be generated for all stores in section 4, this has been omitted for RDF4j. The same queries that have been executed on the base dataset were executed again in order to make it possible to compare the results for the differently scaled datasets with each other. Due to the fact that RDF4j is the only store that has been benchmarked with differently scaled datasets the query results can not be compared to other stores and therefore, the only possible comparison is between the differently scaled datasets. Thus the queries have not been changed. Furthermore the result sets for the different queries did not change when the dataset was scaled. The execution time for each scaling, including the base dataset is shown in table 7.

Query	**Base Dataset**	**5 Hops**	**7 Hops**
1 = P1/P2	57	69	73
2 = P1| P2	240	234	228
3 = (P1| P2)/P3	79	79	85
4 = P1+/P2	78	78	87
5 = ^P1	290	78	382
6 = P1*	247	247	262
7 = P1*| P2*| P3*	194	194	271
8 = P1+/P2*	71	72	113

Table 7: The average execution time per query for RDF4j for the different scalings

As can be seen in table 7, the average execution time does not differ much for queries 1 2 3 4 and 6 for the different scalings. Query 2 was even a bit faster the more data was loaded. Query 7 and 8 had a much higher execution time for 7 hops which is most likely due to the amount of data. Finally query 5 has the lowest execution time for 5 hops and the highest execution time for 7 hops. This observation contradicts the expectation that the execution time would get higher the more data was loaded into the store. The exact reason would require a deeper analysis how RDF4j works. Due to time limitations this analysis is left over for future work.

In figure 7 the distribution of execution times for each query in each scaling is presented. The execution times of query 1 3 and 4 look very similar to each other, which is due to the fact that they do not differ very much for each scaling. The execution times for query 2 show that the the maximum for the highest scaling is much lower than for the other scalings even though the average execution times are rather similar.

The execution times for query 5 look very differently for each scaling. For the base dataset they are distributed over a wide range go from 154ms up to 413ms with the interquartile range covering almost the entire range. This means that the execution times are equally distributed. The range of execution times for the scaling of 5 hops is very small and goes from 34ms to 105ms with an interquartile range covering almost all of it. This means that the execution times are much lower in general for this scaling than for the base dataset. This is also denoted by the average execution times in table 7. The execution times for the scaling of 7 hops goes from 293ms to 685ms but have an interquartile range from 297ms to 413ms, which means that even though the maximum execution time has been deleted from the results to prevent outliers, outliers still exist for this query. Furthermore it shows that the execution times for this scalings are higher than for the other two scalings.

The execution times for query 6 look similar for the base dataset and 5 hops but the interquartile range is smaller and starts higher for the scaling of 7 hops. Furthermore the maximal execution time for 7 hops is lower than for the other 2 scalings. The execution times for the base dataset and 5 hops for query 7 look similar to each other. The interquartile range of the execution times for 7 hops is similar to the other scalings but starts higher. This means that the execution times are higher for 7 hops. Finally the execution times for the first two scalings look similar to each other for query 8, but the execution times for 7 hops have a smaller but higher interquartile range.

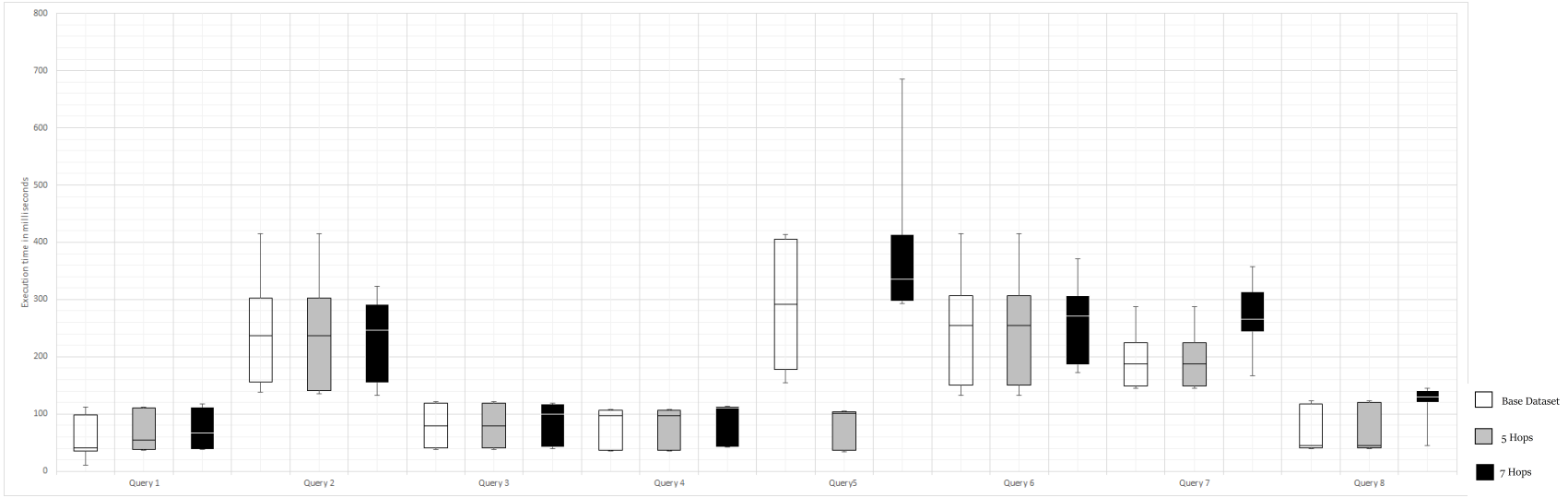

Figure 7: Boxplot for the execution time of RDF4j on the BTC dataset in each of the three scalings

6 Conclusion and Future Research

In order to test how well different RDF stores support SPARQL Property Paths a benchmark for Property Paths has been introduced in this work. To do so quality criteria for good benchmarks have been introduced, for instance the syntactical correctness of queries and data, the capability of scaling the dataset and the representativity of queries and data. Based on these quality criteria existing benchmarks for RDF stores have been reviewed. No benchmarked that was reviewed fulfilled all defined quality criteria to full extent. Although the scalability criterion was fulfilled by most benchmarks, those benchmarks had no representative datasets but synthetic datasets, which are easy to scale. Furthermore tt has been stated that no benchmark that tests the evaluation of Property Path expressions exists.

Based on the reviewed RDF benchmarks the new Property Path Benchmark has been developed. It has been asserted that this new benchmark fulfils the prior defined quality criteria. Even though the representativity was fulfilled for the dataset of the benchmark, it has not been possible for the queries because the queries were designed to test Property Paths and therefore, are not frequently used queries. The Property Patch Benchmark solves challenge of using a representative and scalable dataset by using the Billion Triple Challenge dataset. This dataset consists of real world RDF data and therefore is representative. Furthermore, it can be scaled by the number of iterations the dataset was mined with. Finally the Property Path Benchmark tests all Property Path expressions.

The second contribution of this work are the benchmark results. The benchmarked stores, Virtuoso, Jena, Allegrograph and RDF4j evaluate queries containing Property Path expression differently well. Jena could not return results for any query besides query 1 in under 1 hour. Furthermore the * could not be evaluated at all and the inverse operator returned empty result sets. Virtuoso had also major problems with the inverse operator and returned mostly incomplete results and in one case even a wrong result. RDF4j could handle most Property Path expressions but had problems with some queries containing the * operator. Still most of the results of RDF4j were complete and sound. Allegrograph returned complete and sound result sets and therefore handled all Property Path expressions well. On the other hand when RDF4j was benchmarked with the same dataset as Allegrograph, it also returned complete and sound results and was even faster than Allegrograph.

It can be summarized that even though RDF stores claim to support Property Paths, this is not necessarily the case and that some implementations of Property Paths do not return correct results.

For future research the benchmark should be used to benchmark ample different stores in addition to the stores benchmarked in the context of this work. Furthermore, it can be investigated why the average execution time for the second scaling for RDF4j is much lower than for the base dataset. Finally, Saleem et al. introduced the Linked SPARQL Queries Dataset (LSQ) which provides representative queries from different public SPARQL endpoints [24]. These queries could be used to generate queries which are representative and test property paths and thus, fulfil the representativity criterion for the queries to a full extent.

References

[1] M. Arenas and J. Perez, "Federation and navigation in sparql 1.1," in *Reasoning Web. Semantic Technologies for Advanced Query Answering*, ser. Lecture Notes in Computer Science, T. Eiter and T. Krennwallner, Eds. Springer Berlin Heidelberg, 2012, vol. 7487, pp. 78–111. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33158-9_3

[2] B. DuCharme, *Learning SPARQL, Chapter 2 pp 19-44, Chapter 3 pp 45-100.* O'Reilly Media, Inc., 2011.

[3] E. V. Kostylev, J. L. Reutter, M. Romero, and D. Vrgoč, *The Semantic Web - ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I.* Cham: Springer International Publishing, 2015, ch. SPARQL with Property Paths, pp. 3–18. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25007-6_1

[4] E. Prud'hommeaux, S. Harris, and A. Seaborne, "SPARQL 1.1 Query Language," W3C, W3C Recommendation, 2013. [Online]. Available: http://www.w3.org/TR/sparql11-query/

[5] K. Lee and L. Liu, "Scaling queries over big rdf graphs with semantic hash partitioning," *Proc. VLDB Endow.*, vol. 6, no. 14, pp. 1894–1905, Sep. 2013. [Online]. Available: http://dx.doi.org/10.14778/2556549.2556571

[6] K. Huppler, "The art of building a good benchmark," in *Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. The Art of Building a Good Benchmark, pp. 18–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10424-4_3

[7] J. Gray, *Benchmark Handbook: For Database and Transaction Processing Systems Chapter 1.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[8] A. Averbuch, E. Daskalaki, G. Flouris, and N. Martinez, "Overview and analysis of existing benchmark frameworks," LDBC, Tech. Rep., March 2013, iST: FP7 - 317548. [Online]. Available: http://ldbcouncil.org/sites/default/files/LDBC_D1.1.1.pdf

[9] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea, "Apples and oranges: A comparison of rdf benchmarks and real rdf datasets," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: ACM, 2011, pp. 145–156. [Online]. Available: http://doi.acm.org/10.1145/1989323.1989340

[10] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "Sp2bench: A SPARQL performance benchmark," *CoRR*, vol. abs/0806.4627, 2008. [Online]. Available: http://arxiv.org/abs/0806.4627

[11] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo, "Dbpedia sparql benchmark: Performance assessment with real queries on real data," in *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ser. ISWC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 454–469. [Online]. Available: http://dl.acm.org/citation.cfm?id=2063016.2063046

[12] C. Bizer and A. Schultz, "The berlin sparql benchmark," *International Journal On Semantic Web and Information Systems*, 2009.

[13] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, 2005. [Online]. Available: http://www.websemanticsjournal.org/index.php/ps/article/view/70

[14] P. Boncz and M.-D. Pham, "Social network intelligence benchmark," CIW, Tech. Rep., 2013. [Online]. Available: https://www.w3.org/wiki/Social_Network_Intelligence_BenchMark

[15] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee, *The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I.* Cham: Springer International Publishing, 2014, ch. Diversified Stress Testing of RDF Data Management Systemss, pp. 197–212. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11964-9_13

[16] V. Kotsev, A. Kiryakov, I. Fundulaki, and V. Alexiev, "Ldbc semantic publishing benchmark (spb) - v2.0 first public draft release," LDBC, Tech. Rep., 2014. [Online]. Available: https://github.com/ldbc/ldbc_spb_bm_2.0/blob/master/doc

[17] O. Görlitz, M. Thimm, and S. Staab, "Splodge: Systematic generation of sparql benchmark queries for linked open data," in *The Semantic Web - ISWC 2012*, ser. Lecture Notes in Computer Science, P. Cudrè-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds. Springer Berlin Heidelberg, 2012, vol. 7649, pp. 116–132. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35176-1_8

[18] D. A. Bader, J. Feo, J. Gilbert, J. Kepner, D. Koester, E. Loh, K. Madduri, B. Mann, T. Meuse, and E. Robinson, "Hpc scalable graph analysis benchmark," HPC, Tech. Rep., 2009.

[19] M. Ciglan, A. Averbuch, and L. Hluchy, "Benchmarking traversal operations over graph databases," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, April 2012, pp. 186–189.

[20] J. Gray, *Benchmark Handbook: For Database and Transaction Processing Systems Chapter 11.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[21] T. Käfer and A. Harth, "Billion Triples Challenge data set," Downloaded from http://km.aifb.kit.edu/projects/btc-2014/, 2014.

[22] T. Käfer, J. Umbrich, A. Hogan, and A. Polleres, "Towards a dynamic linked data observatory," in *In LDOW at WWW*, 2012.

[23] M. Arenas, S. Conca, and J. Pérez, "Counting beyond a yottabyte, or how sparql 1.1 property paths will prevent adoption of the standard," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 629–638. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187922

[24] M. Saleem, I. Ali, A. Hogan, Q. Mehmood, and A.-C. Ngonga Ngomo, "The lsq dataset: Querying for queries," in *International Semantic Web Conference (ISWC)*, 2015. [Online]. Available: http://svn.aksw.org/papers/2015/ISWC-PD-LSQ/public.pdf

A Query Mix with Reference Results for BTC dataset

```
1 SELECT ?o1 WHERE{
2   <http://www.w3.org/ns/sparql-service-description#endpoint>
3    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>/
4    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
5    ?o1
6 }
```

Listing 18: Query 1 of the fixed query mix

```
1 <http://www.w3.org/2000/01/rdf-schema#Class>
2 "RDF Property"
```

Listing 19: Reference results for query 1

```
1 SELECT ?o1 WHERE{
2   <http://purl.org/dc/terms/created>
3   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>|
4   <http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
5    ?o1
6 }
```

Listing 20: Query 2 of the fixed query mix

```
1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
2 <http://www.w3.org/2002/07/owl#AnnotationProperty>
3 <http://www.w3.org2002/07owl#DatatypeProperty>
4 <http://www.w3.org/2002/07/owl#FunctionalProperty>
5 <http://purl.org/dc/elements/1.1/date>
6 <http://purl.org/dc/terms/date>
```

Listing 21: Reference results for query 2

```
1 SELECT ?o1 WHERE{
2   <http://purl.org/vocab/bio/0.1/concurrentEvent>
3   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>|
4   <http://www.w3.org/2000/01/rdf-schema#subPropertyOf>/
5   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
6   ?o1
7 }
```

Listing 22: Query 3 of the fixed query mix

```
1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
2 <http://www.w3.org/2002/07/owl#ObjectProperty>
3 <http://www.w3.org/2002/07/owl#SymmetricProperty>
```

Listing 23: Reference results for query 3

```
SELECT ?o1 WHERE{
    <nodeID://b63568>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>+/
    <http://www.w3.org/2000/01/rdf-schema#label>
    ?o1
}
```

Listing 24: Query 4 of the fixed query mix.

```
"Propriete"
"Property"
"Class"
"Classe"
"Datatype"
"Property"
```

Listing 25: Reference results for query 4

```
SELECT ?o1 WHERE{
    ?o1
    ^<http://www.openarchives.org/ore/terms/aggregates>
    <http://chroniclingamerica.loc.gov/lccn/sn85038615#title>
}
```

Listing 26: Query 5 of the fixed query mix

```
<http://chroniclingamerica.loc.gov/lccn/sn85038615/1903-01-27/ed-
1#issue>
<http://chroniclingamerica.loc.gov/lccn/sn85038615/1903-01-28/ed-
1#issue>
<http://chroniclingamerica.loc.gov/lccn/sn85038615/1903-01-29/ed-
1#issue>
[...]
```

Listing 27: The first 3 of 4011 results of query 5

```
SELECT ?o1 WHERE{
   <nodeID://b63568>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>*
   ?o1
}
```

Listing 28: Query 6 of the fixed query mix

```
nodeID://b63568
http://www.w3.org/2003/01/geo/wgs84_pos#lat
http://www.w3.org/2002/07/owl#DatatypeProperty
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2002/07/owl#Class
```

Listing 29: Reference results for query 6

```
SELECT ?o1 WHERE{
  <http://purl.org/dc/terms/created>
  <http://www.w3.org/2000/01/rdf-schema#subPropertyOf>*|
  <http://www.w3.org/2000/01/rdf-schema#label>*|
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>*
  ?o1
}
```

Listing 30: Query 7 of the fixed query mix

```
<http://purl.org/dc/terms/created>
<http://purl.org/dc/elements/1.1/date>
<http://purl.org/dc/terms/date>
<http://purl.org/dc/terms/created>
"Date Created"@en
"has creation date"@en
<http://purl.org/dc/terms/created>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://www.w3.org/2002/07/owl#AnnotationProperty>
<http://www.w3.org/2002/07/owl#DatatypeProperty>
<http://www.w3.org/2002/07/owl#FunctionalProperty>
<http://www.w3.org/2000/01/rdf-schema#Class>
"RDF Property"
<http://www.w3.org/2002/07/owl#Class>
```

Listing 31: Reference results for query 7

```
SELECT ?o1 WHERE{
  <http://reference.data.gov.uk/def/intervals
      /intervalContainsIso8601Week>
  <http://www.w3.org/2000/01/rdf-schema#subPropertyOf>+/
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>*
  ?o1
}
```

Listing 32: Query 8 of the fixed query mix

```
<http://reference.data.gov.uk/def/intervals/intervalContainsWeek>
<http://www.w3.org/2002/07/owl#ObjectProperty>
<http://www.w3.org/2000/01/rdf-schema#Class>
"OWL ObjectProperty"
<http://www.w3.org/2002/07/owl#Class>
<http://www.w3.org/2006/time#intervalContains>
<http://www.w3.org/2002/07/owl#TransitiveProperty>
```

Listing 33: Reference results for query 8.

B Query Mix with Reference Results for Polish DBpedia Dump

```
1 SELECT ?o1 WHERE{
2   <http://dbpedia.org/resource/Farmington,_New_Hampshire>
3   <http://dbpedia.org/property/establishedTitle>/
4   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
5   ?o1
6 }
```

Listing 34: Query 1 for the Polish DBpedia dump

```
1 <http://dbpedia.org/class/yago/Abstraction100002137>
```

Listing 35: Reference result set for query 1

```
1 SELECT ?o1 WHERE{
2   <http://dbpedia.org/resource/Antoninus_Pius>
3   <http://dbpedia.org/ontology/deathPlace> |
4   <http://dbpedia.org/ontology/deathDate>
5   ?o1
6 }
```

Listing 36: Query 2 for the Polish DBpedia dump

```
1 <http://dbpedia.org/resource/Lorium>
2 "0161-03-07"
```

Listing 37: Reference result set for query 2

```
1 SELECT ?o1 WHERE{
2   <http://dbpedia.org/resource/Roman_Wallner>
3   (<http://dbpedia.org/ontology/birthPlace> |
4    <http://dbpedia.org/property/currentclub>)/
5   <http://dbpedia.org/ontology/wikiPageRevisionID>
6   ?o1
7 }
```

Listing 38: Query 3 for the Polish DBpedia dump

```
1 "680733992"
2 "681329865"
```

Listing 39: Reference result set for query 3

```
1 SELECT ?o1 WHERE{
2   <http://dbpedia.org/ontology/championInSingleFemale>
3   <http://www.w3.org/2000/01/rdf-schema#subPropertyOf>+/
4   <http://www.w3.org/2000/01/rdf-schema#range>
5   ?o1
6 }
```

Listing 40: Query 4 for the Polish DBpedia dump

```
<http://dbpedia.org/ontology/Athlete>
```

Listing 41: Reference result set for query 4

```
SELECT ?o1 WHERE{
  <http://dbpedia.org/resource/Category:Airports_in_Zambia>
  ^<http://purl.org/dc/terms/subject>
  ?o1
}
```

Listing 42: Query 5 for the Polish DBpedia dump

```
<http://dbpedia.org/resource/Mfuwe_Airport>
<http://dbpedia.org/resource/Harry_Mwanga_
    Nkumbula_International_Airport>
```

Listing 43: Reference result set for query 5

```
SELECT ?o1 WHERE{
  <http://dbpedia.org/resource/Clark_(disambiguation)>
  <http://dbpedia.org/ontology/wikiPageDisambiguates>*
  ?o1
}
```

Listing 44: Query 6 for the Polish DBpedia dump

```
<http://dbpedia.org/resource/Clark_(disambiguation)>
<http://dbpedia.org/resource/Clark>
<http://dbpedia.org/resource/James_Clark>
<http://dbpedia.org/resource/James_Clarke>
<http://dbpedia.org/resource/James_Clarke_(author)>
```

Listing 45: Reference result set for query 6

```
SELECT ?o1 WHERE {
  <http://dbpedia.org/resource/4_Pines_Brewing_Company>
  <http://dbpedia.org/property/location>*|
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>*|
  <http://purl.org/dc/terms/subject>*
  ?o1
}
```

Listing 46: Query 7 for the Polish DBpedia dump

```
<http://dbpedia.org/resource/4_Pines_Brewing_Company>
<http://dbpedia.org/resource/Manly,_New_South_Wales>
"Sydney_CBD"@en
<http://dbpedia.org/ontology/Company>
<http://www.w3.org/2002/07/owl#Class>
<http://dbpedia.org/resource/Category:Beer_
    brewing_companies_based_in_New_South_Wales>
```

Listing 47: Reference result set for query 7

```
SELECT ?o1 WHERE{
  <http://dbpedia.org/resource/Lions_Gate_Entertainment>
  <http://dbpedia.org/property/divisions>+/
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>*
  ?o1
}
```

Listing 48: Query 8 for the Polish DBpedia dump

```
<http://dbpedia.org/resource/Lionsgate_Films>
<http://dbpedia.org/resource/Lionsgate_Premiere>
<http://dbpedia.org/ontology/Company>
<http://www.w3.org/2002/07/owl#Class>
```

Listing 49: Reference result set for query 8