BeSEPPI: Semantic-Based Benchmarking of Property Path Implementations

Adrian Skubella[1]✉, Daniel Janke[1], and Steffen Staab[1,2]

[1] Institute for Web Science and Technologies
Universität Koblenz-Landau, Germany
{skubella,danijank,staab}@uni-koblenz.de
http://west.uni-koblenz.de/
[2] Web and Internet Science Group
University of Southampton, UK
s.r.staab@soton.ac.uk
http://wais.ecs.soton.ac.uk/

Abstract. In 2013 property paths were introduced with the release of SPARQL 1.1. These property paths allow for describing complex queries in a more concise and comprehensive way. The W3C introduced a formal specification of the semantics of property paths, to which implementations should adhere. Most commonly used RDF stores claim to support property paths. In order to give insight into how well current implementations of property paths work we have developed BeSEPPI, a benchmark for the semantic based evaluation of property path implementations. BeSEPPI measures execution times of queries containing property paths and checks whether RDF stores follow the W3Cs semantics by testing the correctness and completeness of query result sets. The results of our benchmark show that only one out of 5 benchmarked RDF stores returns complete and correct result sets for all benchmark queries.

1 Introduction

The SPARQL Protocol and RDF Query Language (SPARQL) is the standard query language for RDF stores. In 2013 property paths were introduced with SPARQL 1.1. Property paths allow for describing paths of arbitrary length in graphs, which cannot be described with a single SPARQL 1.0 query. For instance, all friends of a friend of a friend etc. from a social network cannot be retrieved with a single SPARQL 1.0 query. With property paths the construct `foaf:knows*` could be used to obtain all desired results with a single query. Furthermore, property paths provide a more concise way to formulate queries. A query that should return all friends of a friend in a social network could use the construct `foaf:knows/foaf:knows`.

In [1] it is shown that more and more queries containing property paths are run against the Wikipedia Knowledge Graph. For instance, of all queries scheduled in January 2018, over 20% contained property paths. In order to ensure that queries containing property paths return the same result sets independently of the used RDF store, the W3C released the official semantics of property paths in [2].

The comparison of query execution times is only meaningful, if the result sets are complete and correct. Therefore, we have developed a benchmark for semantic-based evaluation of property path implementations (*BeSEPPI*). BeSEPPI does not only measure the execution times of property path queries, but also provides unit tests to check if the result sets are complete and correct based on the W3Cs semantics (see section 3). Our

benchmark comes with 236 queries and respective reference result sets, testing various semantic aspects of property paths. Thus, BeSEPPI may also be used by RDF store developers as a unit test to analyze their own implementation of property paths.

We used BeSEPPI to evaluate Blazegraph, AllegroGraph, Virtuoso, RDF4J and Apache Jena Fuseki (see section 4). Our evaluation indicates that most RDF stores do not adhere to the W3Cs semantics completely. The original contributions of this paper[3] are:

1. BeSEPPI: A benchmark testing the execution times as well as the result set correctness and completeness of property path queries (see section 3).
2. An extensive evaluation of 5 common RDF stores (see section 4).

2 Preliminaries

In the following, common definitions for RDF, SPARQL and property paths based on [4],[5] and [6] are given in order to define the terminology used in this work.

2.1 Graph

The Resource Description Framework (RDF) [7] is a general-purpose language for representing information in the web. It uses triples to represent the information as directed, labeled graphs. A graphical representation of an RDF dataset is shown in figure 1. For better legibility prefixes can be used to abbreviate IRIs. An example for such a prefix is given by `PREFIX ppb: <http://ppbenchmark.com/>`. This prefix defines that for instance `ppb:B1` actually means `<http://www.ppbenchmark.com/B1>`.

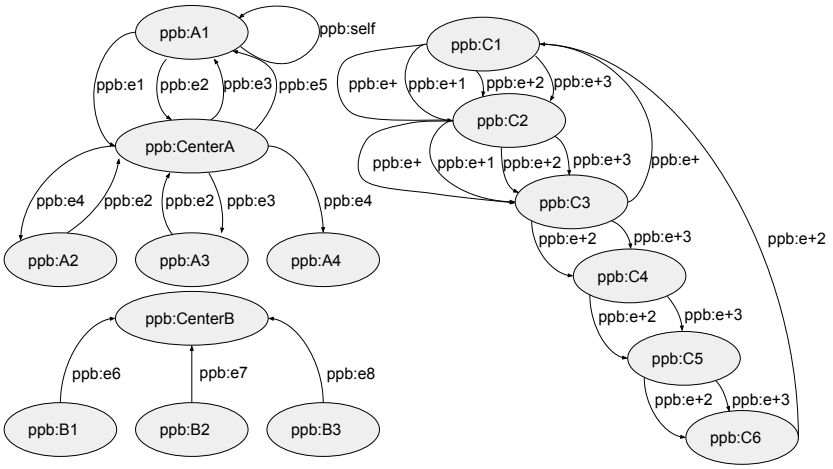

Fig. 1: RDF graphs that are part of BeSEPPI.

Definition 1 (RDF triple).
A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ *is called* RDF triple *where I, L and B are disjoint sets of IRIs, literals and blank nodes, respectively. Furthermore, s is called the* subject*, p the* predicate *and o the* object *of the triple. [4]*

[3] We have presented some preliminary results in a non-archival workshop contribution in [3]. For this paper, we have improved the benchmark by creating a larger variety of queries as well as their correct results sets. These queries are a unit test to check whether property paths implementations adhere to the W3C's semantics.

Definition 2 (RDF graph).
An RDF graph G *is a finite set of RDF triples. Furthermore, the subjects and objects occurring in G are vertices and occurring predicates are edges in G.* $\mathcal{V}_G$ *is the* set of all vertex labels *in G and* $\mathcal{E}_G$ *is the* set of all edge labels *in G.*

Definition 3 (RDF term).
An RDF term *t is an element of* $I \cup L \cup B$. *The* set of all RDF terms *in a graph G is denoted by* T_G.

Definition 4 (Path and Cycle).
A path $P = \ll v_0, e_1, v_1, e_2, v_2, ..., e_n, v_n \gg$ *in an RDF graph G connects two vertices* v_0 *and* v_n *with each other. In a path* v_i *are vertices,* e_i *are edges,* $\forall i, j \in [0, n-1] : i \neq j \Rightarrow v_i \neq v_j$ *and* $\forall i \in [1, n-1] : v_i \neq v_n$. *A path is called* cycle *if* $v_0 = v_n$. *Furthermore, the* path length *is defined by the number of edges between* v_0 *and* v_n.[4]

Example 1: An example for a path between the vertices `ppb:A1` and `ppb:A3` in figure 1 is: $P = \ll$`ppb:A1`, `ppb:e1`, `ppb:CenterA`, `ppb:e3`, `ppb:A3`$\gg$. The length of this path is 2. Moreover, a *self loop* is a cycle of length one. In case of figure 1 the path $P = \ll$`ppb:A1`, `ppb:eSelf`, `ppb:A1`$\gg$ is a self loop.

2.2 SPARQL 1.1 Property Paths

The *SPARQL Protocol and RDF Query Language (SPARQL)* 1.1 is used to query RDF graphs. In the following section the syntax and semantics of the subset of SPARQL 1.1 that is needed for this paper is introduced. The syntax and semantics of property paths are defined following the semantic specification of the W3C in [2].[5]

Syntax

Definition 5 (Property path expression).
A property path expression *can be an* atomic *or a* combined *property path expression.*
Atomic property path expressions:
1) $iri \in I$ *is a* simple property path expression.
2) $!(iri_1|...|iri_n|\hat{}iri_{n+1}|...|\hat{}iri_m)$ *with* $iri_1, ...iri_m \in I$ *is the* negated and inverse negated property set.
Combined property path expressions:
3) $\hat{}E$ *with property path expression E, is the* inverse property path expression.
4) E_1/E_2*, with property path expressions* E_1 *and* E_2*, is the* sequence property path expression.
5) $E_1|E_2$*, with property path expressions* E_1 *and* E_2*, is the* alternative property path expression.
6) $E?$ *with property path expression E is the* existential property path expression.
7) $E*$*, with property path expression E, is the* transitive reflexive closure property path expression.
8) $E+$*, with property path expression E, is the* transitive closure property path expression.
9) (E)*, groups the expression E.*

[4] With our definition of paths, we do not allow cycles to appear within a path. We use this definition since the auxiliary function ALP which is used by the transitive and transitive reflexive property path expression in the W3Cs semantics of property paths [2] uses the same definition of paths.

[5] The notation of property path semantics presented in this section, is based on the definitions of property paths in [6].

Example 2: An example for an existential property path expression with the IRI `ppb:e1` is `ppb:e1?`.

Definition 6 (Property path).
A property path *P is defined as* sEo *where* $s \in V \cup I \cup L \cup B$, $o \in V \cup I \cup L \cup B$ *and* E *is a property path expression.*

Example 3: An example for a property path with an existential property path expression and the variable *?o* is *ppb:notExisting ppb:e1? ?o.*

Definition 7 (Property path Query).
If P *is a property path and V' is a set of variables, then* `SELECT` *V'* `WHERE` *{P} and* `SELECT` * `WHERE` *{P} are* SELECT queries.
If P *is a property path, then* `ASK WHERE` *{*P*} is an* ASK query.*[8]*

Example 4: An example of a SELECT property path query with one variable is shown in listing 1.1.

```
PREFIX ppb: <http://www.ppbenchmark.com/>
SELECT ?o WHERE {ppb:notExisting ppb:e1? ?o.}
```

Listing 1.1: Example of a SELECT property path query.

Semantics

Definition 8 (Evaluation of property path expressions). [6]
Γ *denotes a set of vertex labels with* $\Gamma \supseteq \mathcal{V}_G$ *and p,* $iri_1..iri_m \in I$. *The* evaluation $[[E]]_G$ of a property path expression E *over an RDF graph G is a subset of* $(I \cup L \cup B)^2$ *defined as follows:*

$$
\begin{aligned}
&(1)[[p]]^{\Gamma}_G && := \{(s,o)|(s,p,o) \in G\}\\
&(2a)[[!(iri_1|...|iri_n)]]^{\Gamma}_G && := \{(s,o)|(s,p,o) \in G \wedge p \notin \{iri_1,...,iri_n\}\}\\
&(2b)[[!(\hat{}iri_1|...|\hat{}iri_n)]]^{\Gamma}_G && := [[\hat{}\,!(iri_1|...|iri_n)]]^{\Gamma}_G\\
&(2c)[[!(iri_1|...|iri_n|\hat{}iri_{n+1}|...|\hat{}iri_m)]]^{\Gamma}_G && := [[!(iri_1|...|iri_n)]]^{\Gamma}_G \cup [[(\hat{}iri_{n+1}|...|\hat{}iri_m)]]^{\Gamma}_G\\
&(3)[[\hat{}E]]^{\Gamma}_G && := \{(s,o)|(o,s) \in [[E]]^{\Gamma}_G\}\\
&(4)[[E_1/E_2]]^{\Gamma}_G && := \{(s,o)|\exists r : (s,r) \in [[E_1]]^{\Gamma}_G \wedge (r,o) \in [[E_2]]^{\Gamma}_G\}\\
&(5)[[E_1|E_2]]^{\Gamma}_G && := [[E_1]]^{\Gamma}_G \cup [[E_2]]^{\Gamma}_G\\
&(6)[[E?]]^{\Gamma}_G && := [[E]]^{\Gamma}_G \cup \{(a,a)|a \in \Gamma\}\\
&(7)[[E+]]^{\Gamma}_G && := \bigcup_{i=1}^{\infty} [[\underbrace{E/E/.../E}_{i \text{ times}}]]^{\Gamma}_G\\
&(8)[[E*]]^{\Gamma}_G && := [[E+]]^{\Gamma}_G \cup [[E?]]^{\Gamma}_G\\
&(9)[[(E)]]^{\Gamma}_G && := [[E]]^{\Gamma}_G
\end{aligned}
$$

[6] In [6] the evaluation of the existential property path expression and the transitive reflexive closure property path expression are defined slightly differently from the definition of the W3C in [2]. We have contacted members of the SPARQL working group in order to resolve these differences [9] [10].

Example 5: Assume the existential property path expression *ppb:e1?*, the RDF graph G depicted in figure 1 and $\Gamma = \{$*ppb:notExisting*$\} \cup \mathcal{V}_G$. The evaluation R of the property path expression is : R = $[[$*ppb:e1?*$]]_G^\Gamma$ = $[[$*ppb:e1*$]]_G^\Gamma \cup \{(a,a)|a \in \Gamma\}$ = {*(ppb:A1, ppb:centerA)*} $\cup \{(ppb:A1, ppb:A1), (ppb:A2, ppb:A2), ...\} \cup$ {*(ppb:notExisting, ppb:notExisting)*}. The first set of the union is the evaluation of $[[$*ppb:e1*$]]_G^\Gamma$. The second set denotes all tuples of vertex labels in G and the third part denotes the tuple of the element that was included in Γ additionally to $\mathcal{V}_G$.

In order to obtain information from an RDF store, elements of Γ are bound to variables. These bindings are called *variable bindings*.

Definition 9 (Variable bindings).
The partial function $\mu : V \rightarrow T$ with variables V and RDF terms T, is called a variable binding. *The* domain $dom(\mu)$ *of a variable binding μ is the set of variables on which μ is defined.*

Definition 10 (Evaluation of property paths).
For constants $s \in I \cup B \cup L$, $o \in I \cup B \cup L$ and variables $v, v_1, v_2 \in V$ the evaluation of property paths *is defined as:*

$$(1)[[sEo]]_G := \begin{cases} \{\{\}\}, if(s,o) \in [[E]]_G^\Gamma \text{ where } \Gamma = \mathcal{V}_G \cup \{s,o\} \\ \{\}, else \end{cases}$$

$$(2)[[sEv]]_G := \{\mu|(s,\mu(v)) \in [[E]]_G^\Gamma \wedge dom(\mu) = \{v\} \text{ where } \Gamma = \mathcal{V}_G \cup \{s\}\}$$

$$(3)[[vEo]]_G := \{\mu|(\mu(v),o) \in [[E]]_G^\Gamma \wedge dom(\mu) = \{v\} \text{ where } \Gamma = \mathcal{V}_G \cup \{o\}\}$$

$$(4)[[v_1Ev_2]]_G := \{\mu|(\mu(v_1),\mu(v_2)) \in [[E]]_G^\Gamma \wedge dom(\mu) = \{v_1,v_2\} \text{ where } \Gamma = \mathcal{V}_G\}$$

Example 6: Assume the property path *ppb:notExisting ppb:e1? ?o* where *ppb:notExisting* $\notin \mathcal{V}_G$. Furthermore, assume R from example 5 as the result of the evaluation of the property path expression *ppb:e1?*. According to definition 10 the evaluation of the property path is: [[*ppb:notExisting ppb:e1? ?o*]]$_G$ = $\{\mu_1\}$ with $\mu_1 = \{$*(?o, ppb:notExisting)*$\}$.

Definition 11 (Semantics of SELECT query).
The evaluation $[[Q]]_G$ of a query Q of the form `SELECT` W `WHERE {` P `}` *is the set of all projections $\mu|_W$ of bindings μ from $[[P]]_G$ to W, where the projection of $\mu|_W$ is the binding that coincides with μ on W and is undefined elsewhere.*
The evaluation of `SELECT * WHERE {` P `}` *is equal to the evaluation of* `SELECT W WHERE {` P `}` *where $W = var(P)$ and $var(P)$ denotes the set of all variables in P.*

Definition 12 (Semantics of ASK query). *[11]*
The evaluation $[[Q]]_G$ of a query Q of the form `ASK WHERE {` P `}` *over an RDF graph G is defined as:*

$$[[Q]]_G = \begin{cases} false & if \quad [[P]]_G^\Gamma = \{\} \\ true & otherwise \end{cases}$$

3 Property Path Benchmark BeSEPPI

In order to benchmark the performance of RDF stores with regard to property path queries we introduce our novel benchmark for semantic-based evaluation of property path implementations (BeSEPPI)[7]. BeSEPPI measures the execution times of 236 property

[7] Available as open source under https://github.com/Institute-Web-Science-and-Technologies/BeSEPPI

path queries. These queries are executed on a small dataset that was created for evaluating various aspects of property paths. Furthermore, BeSEPPI comes with reference result sets for each query, which allow for evaluating correctness and completeness of result sets.

3.1 Dataset

The benchmark dataset is a graph consisting of 28 triples. It allows for testing various semantic aspects of each property path expression. The dataset is kept small so that humans can easily create reference result sets for property path queries and evaluate the correctness and completeness of query result sets. The graph is depicted in figure 1.

3.2 Queries

The query set of BeSEPPI consists of 236 queries of which 73 are ASK queries and 163 are SELECT queries. In our benchmark we want to evaluate the performance of each property path expression individually with regard to various semantic aspects. Therefore, we test each expression separately and omit combinations of property path expressions. The queries are organized according to the following 3 dimensions.

Dimension 1: The property path expression.

The first dimension is the property path expression that is tested.

Dimension 2: The number and positions of variables and terms in the query.

According to definition 10 there are 4 possibilities for the number and positions of variables and terms in a query containing a single property path: sEo, sEv, vEo and v_1Ev_2 where s and o are terms v, v_1 and v_2 are variables and E is a property path expression. Queries of the form sEo test for the existence of the path in the dataset and do not return any variable bindings. During our evaluation we have observed that some stores do not support queries with * after the SELECT statement, which do not contain any variables, even though these queries are syntactically correct. Due to the fact that such a query simply returns an empty set if the path in the query does not exist and otherwise an empty variable binding, we have transformed such queries to ASK queries which return `false` or `true`. We expect ASK queries to be supported in all cases whereas SELECT queries with * and without variables have shown to be not supported in some cases.

Dimension 3: Semantic aspects.

Semantic aspects are certain characteristics a query fulfills. Semantic aspects are for instance, that a query returns an empty result set or that the traversed path in the graph has a length of at least 4. Each property path expression has different semantics and therefore, not all semantic aspects can be considered for all property path expressions. Due to the high number of queries in BeSEPPI, describing all queries and the respective semantic aspects is beyond the scope of this paper. In order to still give insight into the query structure we give an overview of queries for each expression and variable-constant combination in table 1. Additionally, we explain two benchmark queries for the existential property path expression in the following section.

Existential Property Path Expression Queries

In order to evaluate the performance of RDF stores for property path queries with the existential property path expression, we use 24 queries. Two exemplary queries and their semantic aspects are presented below. For all queries reference result sets were created to evaluate the correctness and completeness of the result sets returned by the RDF stores.

Expression	$s\mathrm{E}o$	$s\mathrm{E}v$	$v\mathrm{E}o$	$v_1\mathrm{E}v_2$	**Total**
Inverse	6	5	5	4	**20**
Sequence	7	6	6	5	**24**
Alternative	6	6	6	5	**23**
Existential	9	6	6	3	**24**
Transitive Closure	12	9	9	8	**38**
Transitive Reflexive Closure	11	8	8	7	**34**
Negated Property Set	6	5	5	5	**21**
Inverse Negated Property Set	6	5	5	5	**21**
Negated and Inverse Negated Property Set	10	7	7	7	**31**
Total	**73**	**57**	**57**	**49**	**236**

Table 1: Overview of number of queries for each property path expression.

```
PREFIX ppb: <http://www.ppbenchmark.com/>
ASK WHERE {
    ppb:notExisting1 ppb:notExisting2? ppb:notExisting1.}
```

Listing 1.2: Existential property path query where vertices and edge are not existing in the dataset.

In the query shown in listing 1.2 none of the stated IRIs exist in the dataset. According to definition 8 *(ppb:notExisting1, ppb:notExisting1)* $\in [[$*ppb:notExisting2?*$]]_G^\Gamma$. Due to definition 10 the evaluation of the property path in the query is $[[$*ppb:notExisting1 ppb:notExisting2? ppb:notExisting1*$]]_G^\Gamma = \{\{\}\}$. Less formally speaking, this query returns `true` because *ppb:notExisting1* is connected to itself by a path of length zero.

```
PREFIX ppb: <http://www.ppbenchmark.com/>
SELECT * WHERE{
    ppb:A1 ppb:e1? ?o.}
```

Listing 1.3: Existential property path query with existing predicate and one variable.

The query shown in listing 1.3 is of the form sEv. This means that there is one variable in the property path. According to definition 8 (6) the evaluation of the property path expression is: $[[$*ppb:e1*?$]]_G^\Gamma = \{($*ppb:A1, ppb:CenterA*$)\} \cup \{(a, a) | a \in \Gamma\}$. Following definition 10 the evaluation of the property path is $[[$*ppb:A1 ppb:e1? ?o*$]]_G^\Gamma = \{\{?o,$ *ppb:centerA*$\}, \{?o,$ *ppb:A1*$\}\}$. Less formally speaking *ppb:centerA* is returned because the triple *(ppb:A1, ppb:e1, ppb:centerA)* exists in the dataset and *ppb:A1* is returned because a path of length 0 exists between *ppb:A1*.

3.3 Metrics

In order to allow for comparing benchmark results of different stores with each other and to make the results comprehensible, meaningful metrics need to be used. For BeSEPPI we focus on the following metrics.

1. *Query correctness*
   The percentage of correct query results that are returned for each query.
   For SELECT queries: If R_q is the set of all correct results for a query q and $R_q^{\mathcal{S}}$ is the set of returned results of query q executed on RDF store $\mathcal{S}$, then the query correctness is defined as:

$$corr(q) := \begin{cases} 1, & if\ |R_q^{\mathcal{S}}| = 0 \\ \dfrac{|R_q \cap R_q^{\mathcal{S}}|}{|R_q^{\mathcal{S}}|}, & otherwise \end{cases}$$

For ASK queries: If r_q is the correct boolean result for the ASK query and $r_q^{\mathcal{S}}$ is the returned boolean result for an RDF store $\mathcal{S}$, then the correctness is defined as:

$$corr(q) = \begin{cases} 1, & if \quad r_q = r_q^{\mathcal{S}} \\ 0 & otherwise \end{cases}$$

2. *Query completeness*
   The percentage of all possible query results of the query.
   For SELECT queries: If R_q is the set of all correct results for a query q and $R_q^{\mathcal{S}}$ is the set of returned results of query q executed on RDF store $\mathcal{S}$, then the query completeness is defined as:

$$comp(q) := \begin{cases} 1, & if\ |R_q| = 0 \\ \dfrac{|R_q \cap R_q^{\mathcal{S}}|}{|R_q|}, & otherwise \end{cases}$$

   For ASK queries: If r_q is the correct boolean result for the ASK query and $r_q^{\mathcal{S}}$ is the returned boolean result for an RDF store $\mathcal{S}$, then the completeness is defined as:

$$comp(q) = \begin{cases} 1 & if \quad r_q = r_q^{\mathcal{S}} \\ 0 & otherwise \end{cases}$$

3. *Average execution time per query*
   The arithmetic mean $avexec(q)$ of the execution time $t(q)$ of each query q is defined as:
   $avexec(q) = \dfrac{\sum_{i=1}^{n} t_i(q)}{n}$ where n is the number of times a query was executed.

3.4 Execution Strategy

In the first step of the benchmark execution, the complete dataset is loaded into the RDF store that should be benchmarked. Afterwards, all 236 queries are executed once without measuring any metrics in order to warm up the store. After that the 236 queries are executed 10 times and the metrics are measured. The queries are executed one after another and not in parallel. To prevent outliers the highest and lowest execution times are deleted. Finally, the average execution time, the correctness and the completeness are stored in a human readable CSV file.

4 Benchmark Results

In order to evaluate the performance of RDF stores in regard to queries containing property paths we use the property path benchmark BeSEPPI described in section 3.

4.1 Experimental Setting

We benchmarked the property path implementations of 5 common RDF stores, namely Blazegraph 2.1.4[8], AllegroGraph 6.4.1 free edition[9], Virtuoso 7.2 open source edition[10], RDF4J 2.2.4[11] and Apache Jena Fuseki 3.8.0[12]. The RDF stores were benchmarked on an Ubuntu 16.04 machine with 8 GB memory, 500 GB disk space and 4 1.7 Ghz processor cores. The Java version on the machine was 1.8.0.171.

[8] https:/www.blazegraph.com/
[9] https://franz.com/agraph/downloads.lhtml
[10] http://vos.openlinksw.com/owiki/wiki/VOS
[11] http://rdf4j.org/
[12] https://jena.apache.org/documentation/fuseki2/

4.2 Completeness and Correctness

In this section the correctness $corr(q)$ and completeness $comp(q)$ of result sets for each store are presented and it is discussed how the difference between the returned results and the reference result sets might be caused.

Expression \ Store	Blaze-graph				Allegro-Graph				Virtuoso				RDF4J				Jena Fuseki				Total Number of Queries
	Incompl. & Correct	Complete & Incor.	Incompl. & Incor.	Error	Incompl. & Correct	Complete & Incor.	Incompl. & Incor.	Error	Incompl. & Correct	Complete & Incor.	Incompl. & Incor.	Error	Incompl. & Correct	Complete & Incor.	Incompl. & Incor.	Error	Incompl. & Correct	Complete & Incor.	Incompl. & Incor.	Error	
Inverse	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
Sequence	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24
Alternative	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23
Existential	3	0	1	0	6	0	2	0	0	0	0	3	0	0	3	0	0	0	0	0	24
Transitive Closure	0	0	1	0	0	0	0	0	6	0	4	8	0	0	4	0	0	0	0	0	34
Transitive Reflexive Closure	7	0	1	0	5	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	38
Negated Property Set	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	21
Inverse Negated Property Set	0	0	0	0	4	0	6	0	0	0	0	11	0	0	0	11	0	0	0	0	21
Negated and Inverse Negated Property Set	0	0	0	0	6	2	8	0	0	0	0	0	0	0	0	17	0	0	0	0	31
Total	10	0	3	0	24	2	16	0	6	0	4	29	0	0	7	39	0	0	0	0	236

Table 2: Number of queries that returned incomplete, incorrect, or incomplete and incorrect result sets, or threw an error during the execution.

In table 2 an overview of the numbers of queries, which returned only incomplete, only incorrect or incomplete and incorrect result sets, or caused an error during the execution of the query is given. Furthermore, the rightmost column shows the total number of queries for the respective property path expression.

One observation is that all stores return complete, correct and error-free result sets for the inverse, sequence and alternative expressions. A reason for this might be the clarity of their semantics, since their definition is the same in different sources, such as the official SPARQL 1.1 definition, [2] and [6]. Furthermore, the transformation of these property path expressions into SPARQL 1.0 queries is straightforward, such that already implemented SPARQL1.0 query operators could be reused.

In the rest of this section the cases in which queries did not return correct, complete and error-free result sets for each store are discussed.

Blazegraph: Blazegraph returns complete and correct result sets for most queries, but there are 13 result sets which are not complete or correct. The first three queries that did not return complete and correct result sets are ASK queries. These three queries incorrectly returned `true` and have in common that the combination of subject and predicate can be found in the graph whereas the object does not occur. For queries in which also the object occurred in the graph, `true` was correctly returned.

All other queries with incomplete result sets return correct results. The tested property paths of these queries are of the form variable, property path expression, variable. Furthermore, they all involve either the existential or the transitive reflexive closure expression. After examining the missing results, we noticed that only results produced by the term $\{(a,a)|a \in \Gamma\}$ from definition 8 (6) are missing.

AllegroGraph: Our evaluation indicates that the semantics used by AllegroGraph deviates from the W3Cs semantics in case of existential $E?$ and transitive reflexive closure property path expressions $E*$. If $[[E]]_G^{\mathcal{V}_G} \neq \{\}$, AllegroGraph uses the W3Cs semantics. But in cases of $[[E]]_G^{\mathcal{V}_G} = \{\}$, AllegroGraph always returns $\{\}$. Furthermore, if the object and subject are equal in ASK queries the query returns `false` even though `true` would be correct.

AllegroGraph also returns empty result sets, if the negated property set contains at least one non-existing property. Furthermore, if the property path contains two variables, AllegroGraph interprets the inverse negated property set as negated property set, leading to result sets in which the assignments of the two variables to terms are swapped. The same applies to the inverse part of the negated and inverse negated property set.

Virtuoso: Virtuoso does not execute queries with two variables combined with the existential, the transitive closure or the transitive reflexive closure property path expression. For such queries the store returns an error, which says "Transitive start not given". This behavior seems to be a deliberate choice in the design of the RDF store and might have to do with the fact that Virtuoso is built on relational databases. In relational databases very large joins might be necessary in order to answer these queries and therefore, this feature may have not been implemented.

For queries with one or no variable and the existential or transitive reflexive closure property path expression Virtuoso returns complete and correct result sets. For the transitive closure property path expression there are 10 queries, which do not return complete result sets for Virtuoso. These queries all have a cycle like $\ll v_1, e, v_2, ..., v_n, e, v_1 \gg$ as tested semantic aspect and the missing result is always the start vertex v_1 of the cycle. This indicates that the transitive closure property path expression might be implemented in such a way that $[[P*]]_G^{\Gamma}$ is evaluated and the reflexive start is removed from the result set. In such a case, queries with cycles would return correct results except for the starting and the end vertex respectively.

For the negated property set Virtuoso returns correct and complete result sets for each query. For 11 queries with the inverse negated property set Virtuoso returns errors. The queries that return errors are distributed over all query forms and semantic aspects such that we could not identify the underlying cause. Finally, the combination of the negated and inverse negated property set returns complete and correct result sets.

RDF4J: RDF4J returns `false` for three ASK queries with the existential property path expression where the correct result is `true`. In each of the three queries, the subject and object are equal. This indicates, that RDF4J ignores the results included in $\{(a,a) \in \Gamma\}$ in ASK queries with the existential property path expression. Furthermore, RDF4J incorrectly returns `false` as result for ASK queries with a transitive closure property path expression, if they have a cycle as tested aspect.

For queries with the negated property set, the inverse negated property set and the combination of both sets RDF4J does not execute queries of the form subject property path expression object or variable property path object. This means every time such a query is executed the store returns an error.

Apache Jena Fuseki: Fuseki was the only store that executed every benchmark query

without errors and returned complete and correct result sets. It seems that the store follows the W3Cs definition of property path semantics.

4.3 Execution Times

In order to compare the current performances of property path implementations we present and discuss the execution times of the benchmark queries in this section[13]. For this discussion we only take the execution times of queries into consideration for which complete and correct result sets were returned. Out of all 236 queries all stores returned complete and correct result sets for only 134 queries. In figure 2 the sums of $avexec(q)$ of these 134 queries are presented for the individual RDF stores.

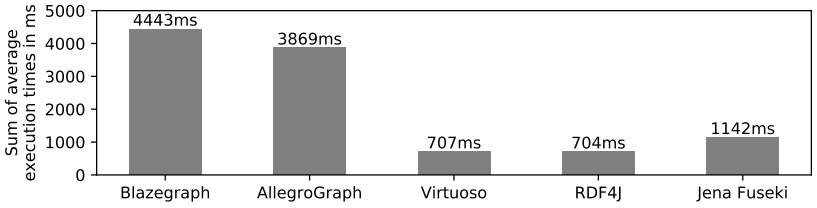

Fig. 2: Sum of average execution times.

When considering all benchmark queries for which complete and correct result sets were returned by all stores, RDF4J and Virtuoso execute queries the fastest. Fuseki was approximately 400ms slower and AllegroGraph took more than 5 times as long as RDF4J or Virtuoso. Blazegraph required 4443ms to execute the queries on average. This means Blazegraph needs more than 7 times longer than Virtuoso or RDF4J.

Expression \ Store	Blazegraph	AllegroGraph	Virtuoso	RDF4J	Fuseki
Inverse	432	515	96	101	170
Sequence	495	595	125	124	235
Alternative	527	561	112	114	186
Existential	428	334	60	56	84
Transitive Reflexive Closure	976	647	109	99	154
Reflexive Closure	1268	850	121	119	176
Negated Property Set	167	186	43	48	66
Inverse Negated Property Set	61	79	15	19	33
Negated and Inverse Negated Property Set	89	102	26	24	38

Table 3: Total execution times in ms for each property path expression.

In table 3 the sums of $avexec(q)$ are shown for queries containing the different property path expressions. For these execution times, the 134 queries were considered, for which complete and correct result sets were returned by all stores. When investigating the influence of the property path expressions dimension on the execution times, table 3 shows that Blazegraph executes queries containing the inverse, sequence, alternative property path expression or any form of negated property sets faster than AllegroGraph, but AllegroGraph is faster when it comes to queries with the existential, transitive or reflexive transitive closure property path expression. Regardless of this, both stores are slower than the other 3 stores for each property path expression.
Virtuoso and RDF4J are the fastest stores for each property path expression and the

[13] A list of all execution times can be found under:
https://github.com/Institute-Web-Science-and-Technologies/BeSEPPI/tree/master/results

differences between the execution times of these two stores are very small. Virtuoso is faster than RDF4j in 4 cases and RDF4J is faster in 5 cases. Therefore, it can be said that both stores have shown a very similar performance in our benchmark. Fuseki averagely needs 1.6 times longer to execute queries than Virtuoso. RDF4J needs at most 54% of the time Blazegraph or AllegroGraph need.
In order to compare how the second dimension of queries, which is the number and position of variables in a query, affect the execution times, we compared the average execution times of all RDF stores for each combination of property path expression and variable position in a query. For instance, in figure 3 the average execution times of queries containing the transitive reflexive property path expression are shown for the different numbers and positions of variables in a query. Due to the fact that Virtuoso does not execute queries containing two variables and the reflexive transitive closure property path expression, the respective bar is missing in figure 3.

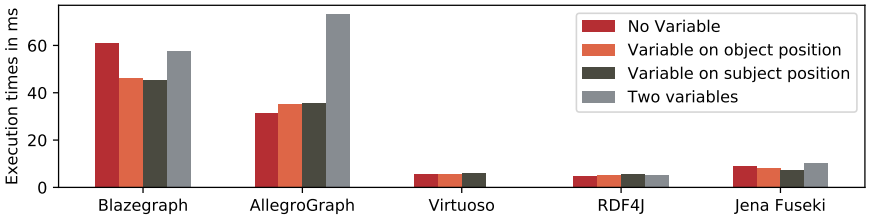

Fig. 3: Average execution times for queries containing the transitive reflexive closure property path expression.

The plots in figure 3 show that Blazegraph processes queries with one variable faster than queries with no or two variables. Contrary to this we expected that queries with two variables would need the most time for execution, since these queries may consider every vertex in the graph as potential start vertex. AllegroGraph executes queries with no variables slightly faster than queries with one variable and queries with one variable faster than queries with two variables. There is nearly no difference in execution times for Virtuoso and RDF4J. Both stores execute the queries faster than the other three stores and the execution time does not increase with the number of variables. Note that Virtuoso does not execute queries with two variables and the transitive reflexive property path expression and therefore, nothing can be said about the execution times for respective queries. Finally Fuseki executes queries with one variable slightly faster than queries with no or two variables. We have expected that queries with two variables would have the highest execution times, but this was only the case for AllegroGraph and Fuseki. Due to the fact that the increase of execution time might depend on the size of the dataset we will test larger datasets in the future.

When investigating the influence of the semantic aspects dimension on the query execution time, most results were not surprising. For instance, transitive closure property path expressions that match with longer paths take longer to be executed. In [12] it is stated that query logs of some public SPARQL Endpoints contain a lot of queries, which return empty result sets. Therefore, it might be beneficial for RDF stores if they can identify these queries quickly to reduce the workload of the store. When focusing on the benchmark queries that return an empty result set, we could figure out that their execution times were similar to the execution times of comparable queries that returned non-empty result sets. Virtuoso and RDF4J have the fastest query execution times (averagely 24ms and 23ms) for queries with empty result sets and average execution times of 23ms and

24ms respectively for comparable queries with non-empty result sets. Blazegraph and AllegroGraph have the longest execution times (averagely 146ms and 114ms) for queries returning empty result sets and take 143ms and 121ms on average for comparable queries with non-empty result sets. Fuseki averagely required 44ms and 45ms for queries with empty and non-empty result sets respectively. This outcome indicates that there is a potential to improve the performance of these stores for queries with empty result sets.

4.4 Summary of Results

In summary, all stores returned complete and correct result sets for queries with an inverse, sequence or alternative property path expression. For queries containing an existential property path expression in it, Blazegraph, AllegroGraph and RDF4J all handle the term $\{(a, a)|a \in \Gamma\}$ differently and are not following the W3Cs semantics. In case of transitive closure property path expressions, Virtuoso and RDF4J ignore results from cyclic paths. AllegroGraph returns empty result sets for queries with the negated property set, if one of the IRIs in the negated property set does not exist in the dataset. Furthermore, AllegroGraph seems to interpret the inverse negated property set as negated property set in queries with two variables. Virtuoso throws errors for ample queries with the inverse negated property set and RDF4J does not execute queries with the negated property set, inverse negated property set or the combination of both sets, where the object of the property path is an RDF term.

Furthermore, Virtuoso does not allow queries with variable path length without a fixed starting or ending point. This means whenever a query with 2 variables containing an existential, a transitive closure or a transitive reflexive closure property path expression is executed, Virtuoso returns an error. From the tested 5 RDF stores only Apache Jena Fuseki could return complete and correct result sets for all queries.

When comparing the execution times of queries for which all stores returned complete and correct result sets, RDF4J and Virtuoso are the two fastest stores in our evaluation. Fuseki needs averagely 60% more time to execute queries than RDF4J and Virtuoso. Blazegraph and AllegroGraph need averagely 260% more time than Fuseki. Furthermore, we have expected that queries with two variables would have the highest execution time for each store, but this was only the case for AllegroGraph and Fuseki.

5 Related Work

Common benchmarks for RDF stores like the Lehigh University Benchmark [13], the DBPedia SPARQL Benchmark [14] or the Berlin SPARQL Benchmark [15] are designed to test the performance of RDF stores in different application scenarios. Since they were created before the release of SPARQL 1.1 they do not test property paths. Furthermore, the Lehigh University Benchmark is the only benchmark that also evaluates completeness and correctness of result sets.

In [16] Gubichev et al. propose an indexing approach called FERRARI to efficiently evaluate property paths. In order to show the efficiency of their approach they also propose a small benchmark with 6 queries over the YAGO2[17] RDF dataset. Although this approach tests queries with property paths, it only measures execution times and does not evaluate correctness or completeness of result sets. In spite of the fact that the benchmark proposed in [18] is not a benchmark for property paths in particular rather than a benchmark primarily designed for streaming RDF/SPARQL engines it tests property paths among various other SPARQL 1.1 features. Even though the completeness and correctness of result sets is not calculated, the results of the benchmark show that most of the benchmarked stream processing systems do not support property path queries.

In [19] a system is presented that generates small datasets based on given queries, their query features (e.g., the `OPTIONAL` or `FILTER` construct) and a data set. Additionally to the small datasets, the system returns the reference result sets for the given queries. They allow for checking the completeness and correctness of the query result sets returned from the evaluated RDF stores. This system is not a benchmark in particular but could be used to create datasets for benchmarks, which evaluate the completeness and correctness of result sets.

In [3] a benchmark for the evaluation of property path support is introduced. This benchmark can use an arbitrary RDF dataset as benchmark dataset and creates queries based on 8 query templates. Due to the small number of queries and the fact, that these queries do not test all property path expressions, this benchmark cannot be used for the semantic evaluation of property path implementations. Nevertheless the results of this benchmark indicate that ample RDF stores return incomplete or incorrect result sets for queries containing property paths.

To the best of our knowledge no RDF benchmark exists that tests if the result sets of property path queries are complete and correct based on the W3Cs semantics.

6 Conclusion

Property paths were introduced with SPARQL 1.1 in 2013. They allow for describing complex queries in a more concise and comprehensive way. In order to evaluate the performances of property path query executions of RDF stores, we have developed a benchmark for semantic-based evaluation of property path implementations called BeSEPPI. BeSEPPI comes with a small RDF dataset especially created for the evaluation of property path queries and 236 queries, which test each property path expression. Our benchmark measures execution times of queries and allows for comparing different RDF stores with each other. Furthermore, BeSEPPI tests if the result sets of the benchmark queries adhere to the W3Cs semantics of property paths and calculates completeness and correctness of result sets.

With BeSEPPI we have benchmarked 5 common stores, namely Blazegraph, AllegroGraph, Virtuoso, RDF4J and Apache Jena Fuseki. The results of BeSEPPI show that only Apache Jena Fuseki could return complete and correct result sets for all 236 queries. Each of the other 4 stores returned incomplete or incorrect result sets for some queries and Virtuoso and RDF4J do not support all types of queries. Furthermore, we have compared the execution times of queries, for which all stores returned complete and correct result sets. This comparison shows that Virtuoso and RDF4J have the lowest execution times. Fuseki is slightly slower. Blazegraph and AllegroGraph needed the most time for the execution of queries containing property path expressions.

With our evaluation we could observe that ample RDF stores do not completely adhere to the W3Cs semantics of property paths. Therefore, BeSEPPI seems to be useful for RDF store developers to evaluate or improve their property path implementations.
The results in [3] have shown, that the correctness and completeness of property path query result sets may depend on the size of the loaded dataset. Therefore, we will perform a semantic evaluation of property path implementations on a large dataset in the future. Furthermore, we will evaluate the correct associativity (i.e.$[[E_1/E_2/E_3]]_G^\Gamma = [[(E_1/E_2)/E_3]]_G^\Gamma = [[E_1/(E_2/E_3)]]_G^\Gamma$) and the correct precedence (i.e.$[[E_1|E_2*]]_G^\Gamma = [[E_1/(E_2*)]]_G^\Gamma$) of several combined property path expressions in the future.

References

1. S. Malyshev, M. Krötzsch, L. González, J. Gonsior, and A. Bielefeldt, "Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph," in *Proceedings of the 17th International Semantic Web Conference (ISWC'18)* (D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, eds.), LNCS, Springer, 2018.
2. S. Harris and A. S. https://www.w3.org/TR/sparql11 query/, "Sparql 1.1 query language."
3. D. Janke, A. Skubella, and S. Staab, "Evaluating sparql 1.1 property path support," in *BLINK/NLIWoD3@ISWC*, 2017.
4. M. Arenas and J. Perez, "Federation and navigation in sparql 1.1," in *Reasoning Web. Semantic Technologies for Advanced Query Answering* (T. Eiter and T. Krennwallner, eds.), vol. 7487 of *Lecture Notes in Computer Science*, pp. 78–111, Springer Berlin Heidelberg, 2012.
5. B. DuCharme, *Learning SPARQL, Chapter 2 pp 19-44, Chapter 3 pp 45-100*. O'Reilly Media, Inc., 2011.
6. E. V. Kostylev, J. L. Reutter, M. Romero, and D. Vrgoč, "Sparql with property paths," in *The Semantic Web - ISWC 2015* (M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, K. Thirunarayan, and S. Staab, eds.), (Cham), pp. 3–18, Springer International Publishing, 2015.
7. M. L. Richard Cyganiak, David Wood, "Rdf 1.1 concepts and abstract syntax," tech. rep., W3C Recommendation, 2014.
8. E. Prud'hommeaux and A. S. https://www.w3.org/TR/rdf-sparql query/#ask, "Sparql query language for rdf w3c recommendation," 2008.
9. https://lists.w3.org/Archives/Public/public-sparql dev/2017OctDec/0009.html.
10. https://lists.w3.org/Archives/Public/public-sparql dev/2018JanMar/0004.htmls.
11. https://www.w3.org/2001/sw/DataAccess/rq23/sparql defns.html#defn_ASK.
12. M. Saleem, I. Ali, A. Hogan, Q. Mehmood, and A.-C. Ngonga Ngomo, "Lsq: The linked sparql queries dataset," in *International Semantic Web Conference (ISWC)*, 2015.
13. Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Web Semant.*, vol. 3, pp. 158–182, Oct. 2005.
14. 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*, ISWC'11, (Berlin, Heidelberg), pp. 454–469, Springer-Verlag, 2011.
15. C. Bizer and A. Schultz, "The berlin sparql benchmark," *International Journal On Semantic Web and Information Systems*, 2009.
16. A. Gubichev, S. Bedathur, and S. Seufert, "Sparqling kleene: fast property paths in rdf-3x," 06 2013.
17. J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "Yago2: A spatially and temporally enhanced knowledge base from wikipedia," *Artificial Intelligence*, vol. 194, pp. 28 – 61, 2013. Artificial Intelligence, Wikipedia and Semi-Structured Resources.
18. Y. Zhang, P. M. Duc, O. Corcho, and J.-P. Calbimonte, "Srbench: A streaming rdf/sparql benchmark," in *The Semantic Web – ISWC 2012* (P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, eds.), (Berlin, Heidelberg), pp. 641–657, Springer Berlin Heidelberg, 2012.
19. V. Thost and J. Dolby, "QED: out-of-the-box datasets for SPARQL query evaluation," in *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018.*, 2018.