Top-k Shortest Paths in large typed RDF Datasets Challenge

Ioannis Papadakis[+], Michalis Stefanidakis[*], Phivos Mylonas[*], Brigitte Endres Niggemeyer[±] and Spyridon Kazanas[+]

[+]School of Information Science and Informatics, Ionian University, Ioannou Theotoki 72, 49100, Corfu, Greece

[*]Dept. of Informatics, Ionian University, 7, Tsirigoti Square, 49100, Corfu, Greece

[±]Heidegruen 36, 30179, Hannover, Germany

{papadakis, mistral, fmylonas}@ionio.gr, brigitteen@googlemail.com, s.kazanas@gmail.com

Abstract. Perhaps the most widely appreciated linked data principle is the one that instructs linked data providers to provide useful information using the standards (i.e., RDF and SPARQL). Such information corresponds to patterns expressed as SPARQL queries that are matched against the RDF graph. Until recently, it was not possible to create a pattern without specifying the exact path that would match against the underlying graph. The advent of the SPARQL 1.1 Recommendation introduced property paths as a new graph matching paradigm that allows the employment of Kleene star * (and its variant Kleene plus +) unary operators to build SPARQL queries that are agnostic of the underlying RDF graph structure. In this paper, we present the Top-k Shortest Paths in large typed RDF Datasets Challenge the highlights the key aspects of property path queries that employ the Kleene star operator.

Keywords: SPARQL 1.1, property paths, navigational queries, Kleene star, ESWC 2016

1 Introduction

In the context of SPARQL, queries are patterns that are matched against an RDF graph. Until recently, it was not possible to create a pattern without specifying the exact path of the underlying graph (in terms of real data and/or variables) that would match against the graph. The advent of SPARQL 1.1 introduced property paths as a new graph matching paradigm that allows the employment of Kleene star * (and it's variant +) unary operators to build SPARQL queries that are agnostic of the underlying RDF graph structure. For example, the graph pattern: *:Mike (foaf:knows)+ ?person* . is a SPARQL query that asks for all the persons that know somebody that knows somebody (and so on) that :Mike knows. Thus, property path queries extend the tradi-

tional graph pattern matching expressiveness of SPARQL 1.0 with navigational expressiveness.

The ability to express path patterns that are agnostic of the underlying graph structure is certainly a step forward. Still though, it is impossible to retrieve the actual paths through a property path SPARQL query. This is due to the fact that navigational queries ask for paths of arbitrary length within the RDF graph without specifying the actual graph patterns that constitute the path at query execution time.

Moreover, it is reasonable to assume that an RDF graph contains numerous paths of varying length between two arbitrary nodes. Therefore, another requirement that arises from the inherent navigational expressiveness of property path queries is the provision of results that are ranked according to their path length.

In this paper, we present the *Top-k Shortest Paths in large typed RDF Datasets Challenge* that took place in Heraklion, Crete, from May, 31st to June, 2nd 2016 as part of the Extended Semantic Web Conference - ESWC 2016. The Challenge evolves around the notion of navigational queries expressed as property path queries employing the Kleene star * (and it's variant +) unary operator. The goal of this challenge is to highlight the key features of this special kind of SPARQL queries and accordingly promote research on this field.

The rest of this paper is structured as follows:

2 Motivation for the Challenge

Navigational queries is a well-known type of queries on graph databases, where the corresponding results contain binary relations over the nodes of the graph [1]. The advent of SPARQL 1.1 Recommendation facilitated the creation of such queries in RDF graphs through the notion of property paths, which are defined as possible routes through a graph between two nodes. Property path queries can be seen as regular expressions over paths between two nodes in the RDF graph.

Equally important to the syntax of such queries is the ability of the underlying triplestore engines to fetch results in reasonable time. Soon after the announcement of property paths, important issues were reported dealing with the efficient processing of the deriving queries. More specifically, it was initially required that all paths that satisfy the specified regular expression be enumerated and for each path the terminal nodes had to be reported. Based on this, valid paths containing cycles could lead to non-terminating results [2], [3]. The Recommendation (www.w3.org/TR/sparql11-property-paths/) exhibited quick reflexes and accordingly dealt with this issue by replacing the initial *bag semantics* (i.e., duplicates within results are detained) underpinning property paths with the more agile *set semantics* (i.e., duplicates within results are not detained).

Apart from the theoretical lifting to the definition of such navigational queries expressed through property paths, there is still much to be done concerning their application to real-time, triplestore engines [3], [4]. For example, the following navigational query fails to terminate when addressed to a triplestore containing 10% of

DBpedia's size (The query was addressed to the Jena TDB containing a DBpedia dump that was downloaded from: http://benchmark.dbpedia.org/):

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

PREFIX dbp: <http://dbpedia.org/resource/>

SELECT DISTINCT ?a WHERE {

?a (!a)*/dbo:language dbp:Thai_language

}
```

By nature, this kind of navigational queries tends to yield plenty of results. This is attributed to the fact that it employs Kleene star unary operators. Therefore, GSPARQL [5] emerged as an extension to the formal SPARQL 1.1 Recommendation capable of addressing such requirement by defining a set of filtering symbols adhering to the property path expression. Alternatively, the use of a LIMIT in such queries can control the number of results. However, in the absence of some kind of semantic ordering, semantically important results may be left out of the result set.

In this Challenge, we argue that results corresponding to shorter paths are more important in terms of semantics than results corresponding to longer paths. The current version of SPARQL 1.1 Recommendation does not impose any criteria for ranking the corresponding results. To alleviate this problem, we consider the length of the matched paths as a generic ranking criterion of the results that correspond to a navigational query. By providing this ranking, the user may safely choose a limit knowing that the returned results are the most important.

3 Structure of the Challenge

This Challenge evolves around the development and deployment of systems that return a specified number of ordered paths between two nodes in a given RDF graph. The systems should be able to accept navigational queries represented as SPARQL queries and accordingly return an ordered set of paths.

The organizing committee evaluates each submitted system based on its ability to address the requirements of two Tasks. Each Task is comprised of two parts. The first part is meant to assess the effectiveness of the corresponding system, whereas the second part is meant to assess its scalability in terms of the number of paths that are requested.

Thus, for each one of the Tasks of the Challenge, the Challengers should develop a system that makes use of the following input data:

1. Start node: The first node of every path that will be returned.
2. End node: The last node of every path that will be returned.
3. k: The required number of paths.

4. ppath expression: A property path expression describing the pattern of the required paths.
5. Dataset: The RDF graph containing the start node and end node.

Each Challenger should respond to each Task by providing *k* paths between the Start node and the End node of the Dataset ordered by their length. Such paths should comply with the given *ppath* expression. A path is defined as a sequence of nodes and edges within the RDF graph:

($A,P_1,U_1,P_2,U_2,P_3,U_3,..,U_{n-1},P_n,B$)

More specifically, *A* corresponds to the Start node (i.e., subject), *B* to the End node (i.e., object), P_i to an edge (i.e., predicate) and U_i to an intermediate node. The path length equals to the corresponding number of edges (e.g., the above path length equals to *n*).

After developing a system pertaining to the aforementioned requirements, each Challenger should deploy the system in the context of the specific Tasks outlined in the following section. Essentially, each Task provides a slightly altered set of input data. Prior to the announcement of the actual data of the Challenge, Challengers have the opportunity to test the effectiveness of their system by employing the training data that are published by the organizers to the website of the Challenge (https://bitbucket.org/ipapadakis/eswc2016-challenge/).

4 Tasks

As stated in the previous section, the Challenge is comprised of two Tasks. Both Tasks require the implementation of a system capable of returning an ordered set of paths. Such a system should be able to respond to five input parameters.

4.1 Task1 - part1

The first part of the first Task asks for a certain number (i.e., k) of paths between two nodes (i.e., A and B) of the evaluation dataset (i.e., RDF graph D), ordered by their length. There is no specific pattern such paths should comply to. In SPARQL dialect, such paths should comply to the following property path expression: *!:**.

Practical Example.

The image below depicts the example RDF dataset D1. Each node is a subject or object while each edge is a predicate.

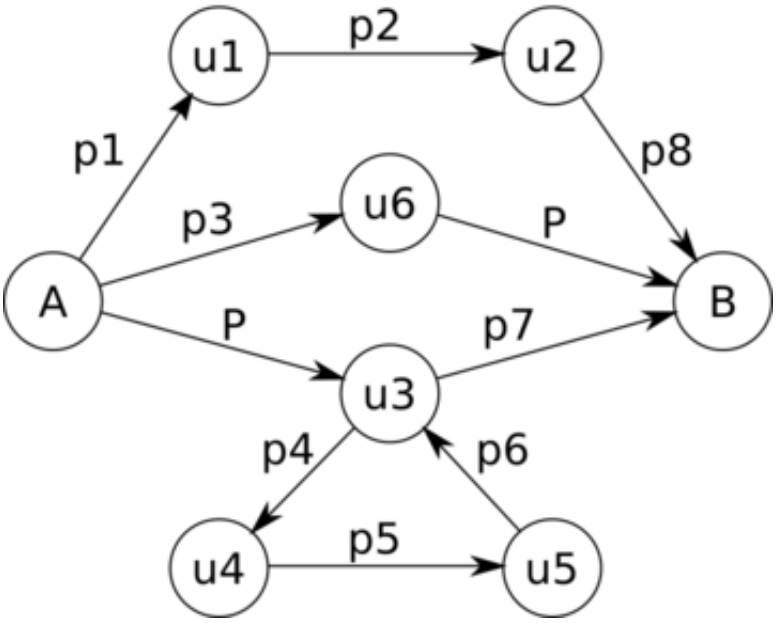

Fig. 1. Example Dataset D1

D1 contains exactly 4 paths from A to B:

1. (A,P,u3,p7,B)
2. (A,p3,u6,P,B)
3. (A,p1,u1,p2,u2,p8,B)
4. (A,P,u3,p4,u4,p5,u5,p6,u3,p7,B)

At this point, it should be mentioned that a path is valid only if it contains unique triples. For example, the path:

(A,P,u3,p4,u4,p5,u5,p6,u3,p4,u4,p5,u5,p6,u3,p7,B)

is not valid, since the triple: *u3,p4,u4* exists more than once.

If task1 requires 3 paths between A and B within D1, the expected results should be the top-3 shortest paths from A to B:

1. (A,P,u3,p7,B)
2. (A,p3,u6,P,B)
3. (A,p1,u1,p2,u2,p8,B)

Note that the first two paths have the same length (i.e., 2), since they both contain two edges). The third path has a length that equals to 3, so it comes after the first two paths.

4.2 Task1 - part2

The second part of the first Task is meant to test the systems implemented by the Challengers in terms of scalability. More specifically, the second part differentiates from the first part by the number of requested paths. This time, such a number is a very large number. Thus, part2 requires a certain number (i.e., *k'*) of paths between two nodes (i.e., *A* and *B*) of the evaluation dataset (i.e., RDF graph *D*), ordered by their length. There is no specific pattern such paths should comply to. In SPARQL dialect, such paths should comply to the following property path expression: *!:**.

4.3 Task2 - part1

The second Task is motivated from the fact that in real life, SPARQL queries usually employ predicates in the context of specific URIs.
Thus, the first part of the second Task differentiates from the corresponding part of the first Task by imposing a specific pattern to the required paths. More specifically, the second Task requires a certain number (i.e., l) of paths between two nodes (i.e., A' and B') of the evaluation dataset (i.e., RDF graph D), ordered by their length. Each and every path should have the edge P as the outgoing edge of A', or the incoming edge of B'. In SPARQL dialect, such paths should comply to the following property path expression: *(P/(!:)*)|((!:)*/P)*.

Practical Example.

Fig. 1 depicts the example RDF dataset D1. Each node is a subject or object while each edge is a predicate. If Task2 requires 2 paths between A and B within D1, the expected results should be the top-2 shortest paths from A to B that have P as their first or last edge:

1. (A,P,u3,p7,B)
2. (A,p3,u6,P,B)

If Task2 requires 3 paths between A and B within D1, the expected results should be the top-3 shortest paths from A to B that have P as their first or last edge:

1. (A,P,u3,p7,B)
2. (A,p3,u6,P,B)
3. (A,P,u3,p4,u4,p5,u5,p6,u3,p7,B)

Note that the path (A,p1,u1,p2,u2,p8,B) is omitted since P is neither the first nor the last edge of the path, so it does not fit into the requirements of the path pattern.

4.4 Task2 - part2

The second part of the second Task is meant to test the solutions provided by the Challengers in terms of scalability. More specifically, the second part differentiates from the first part by the number of requested paths. This time, such a number is a very large number. Thus, Task2 requires a certain number (i.e., *l'*) of paths between two nodes (i.e., *A'* and *B'*) of the evaluation dataset (i.e., RDF graph *D*), ordered by their length. Every path should have the edge *P* as the outgoing edge of *A'*, or the incoming edge of *B'*. In SPARQL dialect, such paths should comply to the following property path expression: *(P/(!:)*)|((!:)*/P)*.

5 Datasets and training material

All systems that participate in the Challenge, should be able to provide ranked lists of paths pertaining to the requirements of each Task. The organizing committee has prepared two datasets for the Challengers, namely the training and the evaluation dataset. The training dataset was published not long after the announcement of the Challenge (available at: https://bitbucket.org/ipapadakis/eswc2016-challenge/downloads) and was accompanied with a set of input parameters together with the expected paths. It was meant to aid Challengers in developing their systems by showing the correct results in a given set of input parameters.

The evaluation dataset (together with the actual input data of each Task) is the actual dataset upon which Challengers should deploy their systems and harvest the paths that are requested from each Task.

The training and the evaluation datasets correspond to transformations of the 10% dataset and the 100% dataset of the DBpedia SPARQL Benchmark (available at: http://aksw.org/Projects/DBPSB.html) respectively. More specifically, both datasets are free of blank nodes and untyped nodes. An untyped node is a node that does not appear as subject in triples adhering to the pattern: "Node rdf:type Type". Scripts are also available to enable the Challengers to build the datasets. The training dataset consists of 9,996,907 triples, while the evaluation dataset consists of 110,621,287 triples (about 11 times larger).

The organizing committee decided to remove untyped nodes from the datasets in an effort to promote the development of systems that make use of some kind of reasoning.

5.1 Description of the training dataset and accompanying material

The training dataset contains 9,996,907 triples and is accompanied by four queries for each Task together with their corresponding results. All the queries are published to the front page of the Challenge's website whereas the corresponding results are published to the *Downloads* section of the website. The results of the training queries are comprised of 38 files that consist of ranked paths. Each file is named after a certain pattern: task_query_<number of paths>.txt. For example, the file task2_q2_3.txt

refers to the second training query of the second task (containing 3 paths ranked by their length.

Each line of every file is a path represented as a json-encoded array, which contains nodes and edges. For example, the following line:
```
["http://dbpedia.org/resource/1952_Winter_Olympics",
"http://dbpedia.org/property/after",
"http://dbpedia.org/resource/1956_Winter_Olympics",
"http://dbpedia.org/property/after",
"http://dbpedia.org/resource/1960_Winter_Olympics",
"http://dbpedia.org/property/officiallyOpenedBy",
"http://dbpedia.org/resource/Richard_Nixon",
"http://dbpedia.org/property/defense",
"http://dbpedia.org/resource/Elliot_Richardson"]
```
corresponds to a path, which starts with node `http://dbpedia.org/resource/1952_Winter_Olympics` and ends with node `http://dbpedia.org/resource/Elliot_Richardson`.

The start node reaches node `http://dbpedia.org/resource/1956_Winter_Olympics` through edge `http://dbpedia.org/property/after`.

Then, node `http://dbpedia.org/resource/1956_Winter_Olympics` reaches node `http://dbpedia.org/resource/1960_Winter_Olympics` through edge `http://dbpedia.org/property/after`.

Then, node `http://dbpedia.org/resource/1960_Winter_Olympics` reaches node `http://dbpedia.org/resource/Richard_Nixon` through edge `http://dbpedia.org/property/officiallyOpenedBy`.

Finally, node `http://dbpedia.org/resource/Richard_Nixon` reaches the final node `http://dbpedia.org/resource/Elliot_Richardson` through edge `http://dbpedia.org/property/defense`.

The length of the aforementioned path equals to 4 (the total number of edges in the path).

6 Assessment process

According to the Challenge's specifications, each Challenger should deploy the system he/she has created in the context of each Task. Moreover, a paper is requested, describing the proposed approach. The Challengers' submissions were assessed according to the following criteria:

6.1 Overall paper quality (50%)

The Program Committee assessed each paper based on well-defined criteria including but not limited to:

- Presentation and organization of the paper
- Scientific / technical quality of the paper
- Technical data of the paper
- English style, writing quality
- Figures and tables
- Innovation of approach
- Efficiency of approach (e.g., by measuring the complexity of an algorithm)
- Consumption of resources
- Scalability

6.2 System quality (50%)

As mentioned earlier in this paper, each Challenger should be able to address the requirements of two Tasks. The Tasks are published to the *Downloads* section of the Challenge's website (i.e., evaluation_input_data.txt) together with the evaluation dataset. The Challengers should address as many Tasks as possible and upload the corresponding result sets of each Task within a specific time frame. The submitted sets are compared against each other in terms of the paths they contain. More specifically, the set that contains the highest number of shortest paths is ranked first, the set that contains the second highest number of shortest paths is ranked second, etc. In case of a tie, the tied sets are compared against the second shortest paths. If the sets remain tied, they are compared against the third shortest paths, and so on. All Tasks have equal weight to the "System quality" criterion. Each Task is assessed separately from the others.

7 Submissions

Finding top-k shortest paths in an RDF graph cannot be easily solved by employing generic graph algorithms, a fact that all contestants to the challenge early recognized and pointed out. The required solutions impose additional constraints, such as loop-free or specially patterned paths. Moreover, the solutions exhibited in the challenge revealed that the performance of any classical graph algorithm can be impractical due to the sheer size of the usually huge and strongly interconnected linked data graphs, which may contain billions -or even trillions- of triples. Dealing with graphs of that size poses also practical problems, as most implementations tend to be memory-based: there is no guarantee that data will always fit in memory. For all these reasons, the proposed solutions decided to use well-known algorithms only as a base, by tweaking or building on top of them.

When preparing a query-answering solution, one can choose to move part of the computational burden to off-line processing that runs once in order to build some kind of index. This pre-built index is consulted later on every query, lowering thus the response time of the answering system. This is the case with the solution proposed in [6]. In their proposal the authors build their modified MinG algorithm inspired from the ρ-index algorithm [9]. The proposed solution uses a hierarchical clustering schema for representing the graph interconnection information divided in levels of graph segments. This schema is built in a step of off-line preprocessing and records the node interconnections in intra-segment Path Type Matrices (PTMs),as well as inter-segment Inverted Files (INFs) for various levels of segmentation. The authors employ compaction tweaks in order to keep the required storage space to reasonable size. The same hierarchical structure is used during querying: beginning from both start and end of the requested path and iterating through the levels of segment information, all valid solutions are accumulated. A final step removes duplicate triples and sorts by path length, in order to achieve the final top-k solution. Due to compaction techniques the proposed implementation fares better for specific graph patterns in terms of indexing space and time compared to the algorithms it is based upon.

In a more graph-theoretical based approach, [7] extends the classical Eppstein's top-k shortest path algorithm [10], which uses a bounded-degree graph to output k paths using breadth-first search. The proposed solution chooses to implement a totally in-line (query-time) processing scheme, which computes a single source shortest path tree T for the target node of any query. The proposed solution augments the classical algorithm's steps by pruning any invalid paths, excluding them from further processing. A further modification allows for the discovery of paths starting or ending with a particular predicate, as requested in challenge terms. The algorithm is symmetrical to any end of the path; in order to find paths ending in a particular predicate the authors invert the graph and treat the ending node as the beginning node. Similarly to other contestant solutions, the authors had to employ memory compaction techniques, such as representing URIs with integer values, in order to ensure that the totality of graph data fit in memory during query processing. The proposed solution complies with the additional requirements of the challenge with a time complexity which is only moderately above the one of Eppstein's generic algorithm. The implementation results show that, in general, the overhead increases with the number of loops in the queried graph. This is why the successful ability to prune invalid paths is essential characteristic of the proposed solution.

The last, but not least, participating solution to the challenge [8] follows a more application-oriented approach. The authors build on an existing platform for automated story-telling [11] that reduces the number of arbitrary resources revealed in interconnecting paths. The main characteristic of the proposed system is a computationally inexpensive and asynchronous server-side interface (Triple Pattern Fragments) that decomposes queries into simple and small parts. The application clients are in charge of querying by requesting and accumulating shortest paths ordered by length, while checking at the same time that the extracted paths fulfill the requirements of the challenge, i.e., they constitute loop-free unique-triple solutions. The implementation favors the querying of shortest paths first, which is beneficial to the control of candidate

path explosion. An additional property of the proposed system is its decentralized nature: decomposing the initial query into small parts allows for the employment of multiple data endpoints without modification of the proposed algorithm, providing thus a solution to memory limitations imposed by a single centralized linked data graph. While more data needs to be buffered and streamed, the implementation results testify that the proposed solution scales very well with the size of input graph.

8 Assessment

According to the aforementioned assessment process, each one of the three Challengers that registered to the event had to submit a paper describing the overall approach. Then, after the publication of the evaluation dataset, each Challenger had to submit a set of four ranked path lists pertaining to the four sets of input parameters (two for each Task) that also have been published to the Challenge's website.

Each paper was assigned for reviewing to three members of the Program Committee. Then, the Challengers had to address the requirements of each Task by uploading the corresponding ranked path lists.

The scoring of each Challenger is outlined in Table 1.

Table 1. Challengers scoring.

Challenger	Task1, part1 (no. of paths)	Task1, part2 (no. of paths)	Task2, part1 (no. of paths)	Task2, part2 (no. of paths)	Paper review
1	377	450	370	4201	7
2	377	53008	374	52664	4
3	10				-2

The Challenger ids at the first column refer to the following teams:

1. Sven Hertling, Markus Schröder, Christian Jilek and Andreas Dengel. Top-k Shortest Paths in Directed, Labeled Multigraphs
2. Zohaib Hassan, Mohammad Abdul Qadir, Muhammad Arshad Islam, Umer Shahzad and Nadeem Akhter. Modified MinG Algorithm to Find Top-K Shortest Paths from large RDF Graphs
3. Laurens De Vocht, Ruben Verborgh, Erik Mannens and Rik Van de Walle. Using Triple Pattern Fragments To Enable Streaming of Top-k Shortest Paths via the Web

More specifically, all Challengers managed to compute some paths according to the requirements of the first part of the first Task. All three path lists were analyzed to see whether they conformed to the following two criteria:

1. Each list contains unique paths
2. Each path contains unique triples

Thus, the first Challenger submitted 383 paths of which 377 were valid, whereas the second Challenger submitted 377 valid paths. The third Challenger submitted 10 valid paths. Consequently, we had a tie between the first two Challengers.

For the remaining tests, the third Challenger failed to submit any results at all, so the team was eliminated from the system evaluation.

For the second part of the first Task, the first Challenger submitted 456 paths of which 450 were valid, whereas the second Challenger submitted 53008 valid paths. Consequently, the second Challenger won this leg of the Challenge.

For the first part of the second Task, the first Challenger submitted 370 valid paths whereas the second Challenger submitted 374 paths of which 370 were valid. Consequently, we had a tie between the first two Challengers.

For the second part of the second Task, the first Challenger submitted 4235 paths of which 4201 were valid, whereas the second Challenger submitted 52664 valid paths. Consequently, the second Challenger won this leg of the Challenge.

Finally, the second Challenger also won the paper reviewing process. Consequently, the second Challenger was the overall winner of Top-k Shortest Paths in large typed RDF Datasets Challenge and was accordingly handed over the prize of the Challenge (i.e., annual subscription to the Commercial License of Blazegraph's graph database with GPU acceleration).

9 Conclusions – Lessons learned

The *Top-k Shortest Paths in large typed RDF Datasets Challenge* was organised for the first time under the auspices of the 13th ESWC 2016 conference. Its primal goal is to raise the awareness of the semantic web community to property paths that have been recently introduced as part of the SPARQL1.1 Recommendations. A property path query that employs the Kleene star (*) or its variant Kleene plus (+) operators deserves much attention, since it exhibits certain scalability issues that are directly proportional to the size of the underlying RDF graph.

Although the participation to the Challenge could be greater, still all three qualified Challengers presented diverse approaches to address the requirements of the Challenge. More specifically, the third Challenger aimed in creating a custom-made index structure that was specifically designed to serve navigational queries in RDF graphs. The first Challenger followed a completely different approach. He employed triple pattern fragments technology in an effort to break the original query down to sequences of triple pattern fragment requests that were accordingly dispatched to clients for asynchronous processing. Finally, the second Challenger modified a well-known path finding algorithm to fit the requirements of the Challenge.

All three Challengers realized that the bigger the RDF graph, the more difficult it is to retrieve the requested results. Thus, scalability issues that are posed by the very nature of navigational queries are identified as a significant issue of this special kind of property path queries. Along these lines, future work is targeted towards the creation of systems capable of efficiently handling such queries in large RDF graphs.

10 References

1. Zhang, X., Van den Bussche, J.: On the power of sparql in expressing navigational queries. The Computer Journal, pp. 128, 2014.
2. Gubichev, A., Bedathur, S. J., Seufert, S.: Sparqling kleene: Fast property paths in rdf-3x. In: First International Workshop on Graph Data Management Experiences and Systems, GRADES '13, pp. 14:1-14:7, New York, NY, USA, 2013. ACM.
3. Arenas, M., Conca, S., Perez, J.: 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, Www '12, pp. 629-638, New York, NY, USA, 2012. ACM
4. Arenas, M., Gutierrez, C., Miranker, D. P., Perez, J., Sequeda, J. F.: Querying semantic data on the web? ACM SIGMOD Record, 41(4):6-17, 2013.
5. Sakr, S., Elnikety, S., He, Y.: G-sparql: a hybrid engine for querying large attributed graphs. In: Proceedings of the 21st ACM international conference on Information and knowledge management, pp. 335-344. ACM, 2012.
6. Hassan, Z., Qadir, M. A., Islam, M. A., Shahzad, U., Akhter, N.: Modified MinG Algorithm to Find Top-K Shortest Paths from large RDF Graphs, ESWC 2016
7. Hertling, S., Schroeder, M., Jilek, C., Dengel, A.: Top-k Shortest Paths in Directed, Labeled Multigraphs, ESWC 2016
8. De Vocht. L., Verborgh. R., Mannens. E. Van de Walle. R.: Using Triple Pattern Fragments To Enable Streaming of Top-k Shortest Paths via the Web, ESWC 2016
   Barton, S.: Indexing graph structured data. Ph.D. Thesis, Masaryk University, Brno, Czech Republic, 2007.
9. Eppstein, D.: Finding the k shortest paths. SIAM Journal on computing, 28(2):652–673, 1998.
10. De Vocht, L., Beecks, C., Verborgh, R., Seidl, T., Mannens, E., Van de Walle, R.: Improving semantic relatedness in paths for storytelling with linked data on the web. In: The Semantic Web: ESWC 2015 Satellite Events - ESWC 2015 Satellite Events Portoroz, Slovenia, May 31 - June 4, 2015, Revised Selected Papers. pp.31–35 (2015)