Using SPARQL – the practitioners' viewpoint

Paul Warren, Paul Mulholland

Knowledge Media Institute, The Open University, U.K.
{paul.warren, paul.mulholland}@open.ac.uk

Abstract. A number of studies have analyzed SPARQL log data to draw conclusions about how SPARQL is being used. To complement this work, a survey of SPARQL users has been undertaken. Whilst confirming some of the conclusions of the previous studies, the current work is able to provide additional insight into how users create SPARQL queries, the difficulties they encounter, and the features they would like to see included in the language. Based on this insight, a number of recommendations are presented to the community. These relate to predicting and avoiding computationally expensive queries; extensions to the language; and extending the search paradigm.

Keywords: SPARQL, user survey, query construction, user difficulties

1 Introduction

Understanding how SPARQL is being used is beneficial for query optimization, for the evolution of the language, and to provide insight into how the language should be taught. The availability of log data from SPARQL endpoints has enabled a number of usage studies. This paper reports on a survey of SPARQL users which complements these previous studies by providing some insight into the user experience. Apart from providing the users' own reported usage of language features, we also report on the users' perceived difficulties, their modes of thinking about queries, and their suggestions for additional features.

Section 2 provides a brief overview of previous related work. Section 3 then provides information on the survey respondents, their tools and the data they query. Section 4 reports on usage patterns, and compares these results to those found from log data analysis. Section 5 reports on the difficulties which people experience, and how that might inhibit the creation of queries. Section 6 reports on the query construction process. Section 7 then reports on two general questions; one relating to additional requirements for the language; the other being an opportunity for respondents to make any final comments. Finally, Section 8 makes some recommendations, specifically for those developing tools and for those thinking about extending the SPARQL language. Note that, in Sections 3 to 7, the various survey questions have been grouped together by themes; the order of presentation here is not exactly as in the survey.

adfa, p. 1, 2011.

2 Previous work

Gallego et al. (2011) analyzed logs from the USEWOD2011 challenge, specifically from DBpedia[1] and Semantic Web Dog Food[2] (SWDF). They concluded that most queries were relatively simple. The majority contained a single triple pattern (66.5% for DBpedia and 97.5% for SWDF), with an exponential decline in number of queries with increasing number of triples. 4.25% of DBpedia queries and 2.19% of SWDF queries used joins. The most common join type was subject-to-subject (SS), at 59.2% of all DBpedia joins and 60.5% of SWDF; subject-to-object (SO), at 35.9% of DBpedia joins and 32.7% of SWDF; and object-to-object (00), at 4.7% of DBpedia joins and 4.5% of SWDF. They noted that chains in 98% of the queries had length one, with the longest chain of subject-to-object triples being of length five.

An emphasis on simplicity, complemented by a long tail of complex queries was also found by Buil-Aranda et al. (2015), who noted that "around 50% of the queries contain just a single triple pattern", and also by Rietveld and Hoekstra (2014). The latter compared an analysis of DBpedia logs with DBpedia queries created using a SPARQL editor, YASGUI. The assumption, based on the work of, e.g. Raghuveer (2012), is that the majority of the former came from applications. Thus, it is possible to compare queries generated by applications with those generated manually. This indicates that the manual queries are more complex, having on average more triple patterns and more joins. Moreover, the manual queries tended to use more SPARQL features, e.g. the LIMIT feature was used by 42% of the manually-generated queries, but only 12% of the DBpedia queries overall. This suggests that the use of SPARQL queries within an application should be seen as somewhat different from the use of SPARQL in interactive mode.

3 The respondents and their environment

Potential respondents were contacted by email, using a variety of mailing lists[3]. There were 53 respondents. The survey used a web survey tool, *Online surveys*[4], and contained 20 questions covering a wide range of the aspects of working with SPARQL. Although respondents were not obliged to answer all questions, almost all the questions were answered by the majority of the respondents, with many questions achieving around 50 responses. When asked for their primary application area, the majority (58%) of the respondents selected 'computer science and information technology' from the list of application areas. 13% indicated 'social sciences and the humanities', whilst 4% indicated each of 'biomedical', 'business and economics', 'engineering' and 'physical sciences. Finally, 15% allocated themselves to the 'other'

[1] http://www.DBpedia.com
[2] http://www.semanticweb.org
[3] semantic-web@w3.org; https://www.w3.org/community/geosemweb; https://www.linkedin.com/groups/{86246, 2002133, 60636, 129217, 38506, 138726, 3063585}; https://groups.google.com/forum/#!forum/semantic_web.
[4] https://www.onlinesurveys.ac.uk/; survey at https://openuniversity.onlinesurveys.ac.uk/sparql-survey-3

category. This included three (6%) who specifically referred to activities related to Wikidata software.

Table 1 illustrates how respondents replied to the question: 'How would you best describe your role?'. The table also gives the percentage breakdown for those respondents who had described their application area as 'computer science and information technology', showing that respondents in this category were under-represented as end-users but over-represented as developers and researchers.

Table 1. Response to question: *how would you best describe your role?*

	%age overall N = 53	**%age CS & IT** N = 30
end-user using SPARQL to query linked data	32%	23%
developer using SPARQL to create end-user applications	30%	37%
researcher into RDF, SPARQL, Linked Data etc.	30%	37%
other	8%	3%

Respondents were then asked whether they used a software client to assist in query building, e.g. to provide auto-completion or enable natural language queries. The question was specifically directed at those who identified themselves as end-users. However, there was no restriction on answering the question, and there were 45 respondents to this question, far more than had identified themselves as end-users. Of these 45, 29% used a software client; 9 respondents used the Wikidata query service.

The majority of respondents had been using SPARQL for some time. Of the 52 responses to the question on how many years they had been working with SPARQL, 44% were in the 'more than 5' category; 13% '4 to 5'; 10% '3 to 4'; 15% in each of the categories '2 to 3' and '1 to 2'; and only 2% responded with 'less than 1'.

Table 2 gives the responses to a question about typical number of triples in the databases being queried. Here, there seems to be a polarization. 50% were divided equally between the highest and lowest categories, with the other 50% divided amongst the remaining categories.

Table 2. Typical number of triples in database being queried (millions) - %age of respondents.

< 1	**1 to 3**	**3 to 10**	**10 to 30**	**30 to 100**	**100 to 300**	**300 to 1,000**	**> 1,000**
25%	8%	10%	8%	12%	2%	12%	25%

In summary, the respondents were relatively experienced in using SPARQL; were end-users, developers and researchers, with a number playing multiple roles; were chiefly users of SPARQL directly, although with some using software clients; and worked with knowledgebases ranging from the quite small to the very large. Although the question was not asked directly, it appears that an appreciable number work with Wikidata.

4 Usage patterns

There were eight questions concerned with the kinds of queries which users create. Some questions asked about frequency of use; these were required to be answered as 'frequently', 'occasionally', or 'never'. In these cases, 0 was assigned to 'never', 0.5 to 'occasionally', 1 to 'frequently', and a composite index was calculated by taking the mean of these data. Where appropriate, the tables show this index, and also the standard deviation calculated from the data.

Triple patterns

Table 3 shows the response to a question about frequency of use of the eight triple patterns. These are the patterns created by the use of either a constant or variable in each of the subject, predicate and object positions, e.g. VCC represents a variable as subject, a URI as predicate and URI or literal as object. In addition, the table shows the results of the analysis which Gallego et al. (2011) conducted with DBpedia and SWDF and the analysis which Rietveld and Hoekstra (2014) conducted with DBpedia logs and YASGUI queries to DBpedia. As pointed out in Section 2, the last of these comes from queries generated manually, whilst the other log data is assumed to be dominated by queries from applications.

Table 3. Frequency of use of triple patterns.

	VCC	**VCV**	**CCV**	**CVV**	**VVC**	**CVC**	**VVV**	**CCC**
Survey responses (N = 50)								
Index, s.d.	0.84,0.3	0.80,0.3	0.77,0.3	0.74,0.4	0.62,0.4	0.51,0.4	0.44,0.4	0.41,0.4
Query log analysis from Gallego et al. (2011) - %age of queries using triple pattern								
DBpedia	7.00%	3.45%	66.35%	21.56%	0.37%	0.20%	0.04%	1.01%
SWDF	46.08%	4.21%	47.79%	0.52%	0.19%	0.006%	1.18%	0.001%
Query log analysis from Rietveld and Hoekstra (2014) - %age of queries using triple pattern								
DBpedia	19.92%	19.06%	42.71%	6.10%	3.92%	0.06%	1.88%	0.17%
YASGUI	45.91%	36.22%	7.10%	2.95%	2.88%	0.12%	1.61%	0.28%

The triples are ordered from left to right in decreasing size of the composite index. Whilst the ranking from the various analyses is not identical, there is clearly some correlation. In particular, there is general agreement on the division between the most used four triples and the least used four. The only exception to this is Gallego et al.'s (2011) SWDF analysis, which gives less prominence to CVV and more to VVV. The 'top' four include only one triple with a variable in the predicate position (CVV).

A Spearman's rank correlation between the survey responses for the top four triples, revealed that there are three significantly correlated pairs [5] (CCV/VCC, CVV/CCV, VCC/VCV) and three pairs which are not significantly correlated (VCV/CVV, VCV/CCV, VCC/CVV). Moreover, there is a significant gap between

[5] At the $p < 0.5$ level; note that, because of the ties, the p-values are not exact. Statistical analysis was undertaken using the R statistical package (R Core Team, 2014).

these two groups. The least significant correlation of the first group (VCC/VCV) has p = 0.028, whereas the highest correlation of the second group (VCV/CVV) has p = 0.192. High correlation between triple pairs could indicate that the two triples are frequently used together in a query. However, it might be that people who use one triple of a pair, also tend to use the other, albeit in different queries. It may be relevant to note that the most correlated pair (VCC and CCV) are triples with only one variable; in one case as subject, in the other case as object.

A k-means cluster analysis of the users, based on their response to this question, with k = 2, gave a breakdown into two approximately equal clusters. In one cluster (which we call *light users*), the triples most used were VCC, VCV and CCV. The other cluster (which we call *power users*) used these triples, but also made considerable use of CVV, VVC, and CVC. Thus, the differentiating factor between the two clusters was the use of a variable as predicate. The majority (59%) of the end-users were in the light user category; the majority (75%) of the developers were in the power user category. Setting k = 3 gave a breakdown into those who used chiefly VCC and VCV; a group who used these two triples, but also made considerable use of CVV, CCV, and to a lesser-extent VVC; and a group who made considerable use of all eight triples. The first group can be characterised by not using a variable in the predicate position. The second group make some use of variable predicates, chiefly CVV and VVC. The third group uses all the triples. Thus, the three groups can be largely characterised by their use of variable predicates. Setting k = 4 leaves the first and last of these three groups little changed, but redistributes the other group into one which extends the repertoire of the first group by the use of CCV and another which extends further by making greater use of CVV and VVC. Once again, the differentiating factor is the use of a variable predicate.

Number of triple patterns per query

Figure 1 shows the responses to a question about how many triple patterns the respondents typically use in a query, and the maximum number of triples they use.

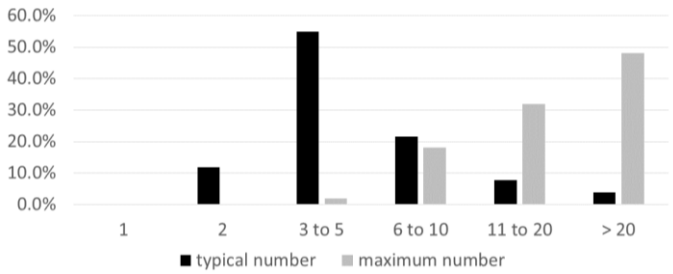

Fig. 1. Typical (N = 51) and maximum (N = 50) number of triple patterns per query - %age respondents by category

This data differs from the log analysis of Gallego et al. (2011) and Rietveld and Hoekstra (2014), which found a large number of queries with only one triple. However, a person who 'typically' uses two or three triples, may still make appreciable

use of one triple queries. Moreover, a study by Bielefeldt et al. (2018) revealed that, of queries generated manually (i.e. not by script), 17.3% contained at most one triple.

The joins

Table 4 shows the response to a question about the kinds of joins respondents use. The joins are ordered by the composite index. The responses indicate a clear distinction between the three most popular and the three least popular. The table also shows log data analyses. That from Gallego et al. (2011) is more indicative of a split into three groups, with OO occupying an intermediate position. That from Rietveld and Hoekstra (2014) gives more prominence to SS and less to SO. In any case, there is clear agreement that joins involving a predicate are not often used. Of this latter category, the most used is SP. This might indicate the use, for example, of a triple with a variable as predicate, and then that same variable occurring as the subject of rdfs:subPropertyOf in another triple.

Table 4. Frequency of use of joins.

	SS	**SO**	**OO**	**SP**	**OP**	**PP**
Survey responses (N = 48)						
Index, s.d.	0.85, 0.3	0.81, 0.3	0.75, 0.4	0.46, 0.4	0.40, 0.4	0.34, 0.3
Query log analysis from Gallego et al. (2011) - %age of queries						
DBpedia	59.23%	35.88%	4.66%	0.19%	0.00%	0.04%
SWDF	60.50%	32.74%	4.46%	2.13%	0.03%	0.13%
Query log analysis from Rietveld and Hoekstra (2014)[6] - %age of queries						
DBpedia	74%	19%	2%	5%	< 1%	< 1%
YASGUI	84%	15%	< 1%	< 1%	< 1%	< 1%

When we compare the usage of joins by the two groups identified previously, there is a clear difference. The power users claim to make more use of each type of join, and this difference is greater for joins involving a predicate. When we compare end-users and developers, the developers also make considerably more use than the end-users of the joins involving a predicate. For the other three joins, there was little difference between the level of usage by end-users and developers.

Query types

Table 5 shows the response to a question about the form of query result respondents use, along with related analysis of log data from Gallego et al. (2011) and from Rietveld and Hoekstra (2014). The data illustrate the complete dominance of SELECT, which is confirmed by the more recent study of Bielefeldt et al. (2018). The survey respondents put more emphasis on CONSTRUCT than is apparent from the log analysis; this might be a consequence of the respondents being self-selecting. On the other hand, the YASGUI data puts greater emphasis on DESCRIBE and ASK.

[6] These data are approximate, having been interpreted from a bar chart.

Table 5. Query types.

	SELECT	CONSTRUCT	DESCRIBE	ASK
Survey responses (N = 50)				
Index, s.d.	0.95, 0.2	0.49, 0.4	0.30, 0.3	0.28, 0.3
Query log analysis from Gallego et al. (2011) - %age of queries				
DBpedia	96.9%	1.5%	0.002%	1.6%
SWDF	99.7%	0.01%	0.002%	0.2%
Query log analysis from Rietveld and Hoekstra (2014) - %age of queries				
DBpedia	96.17%	3.00%	0.56%	0.26%
YASGUI	93.91%	0.72%	1.49%	3.87%

When we compare the usage of the query types by the light and power users, for three of the query types there is little difference between the two groups. The exception is CONSTRUCT, which is much more used by the power users than the light users. Comparing end-users and developers, there was little difference in the use of SELECT and ASK; the former because it was heavily used by both sets of users; the latter because it was little used. There was a very big difference in the use of CONSTRUCT; 76% of the end-users claimed never to use it, whilst for developers this figure was 20%[7]. For DESCRIBE, there was a similar but less exaggerated difference; 59% of the end-users and 27%[7] of the developers never used this query type.

Query features

Respondents were asked which features they used, from the list shown in Table 6. The table also provides data from Gallego et al. (2011) and Rietveld and Hoekstra (2014), showing the percentage of queries containing each of the features. The log analyses did not include all the features covered by the survey; conversely the log analyses included features not covered by the survey. The survey data shows an extensive use of FILTER and OPTIONAL, with some use of the other four features. Whilst MINUS has the lowest index, this was the result of having a far greater proportion of 'never' responses than the other five features; the proportion of 'frequently' responses was slightly greater than for UNION.

When we compare the usage profiles for the light and power users, there is no appreciable difference. BIND and FILTER have very similar indices for both groups; FILTER NOT EXISTS, MINUS and OPTIONAL are slightly less used by the power users than the light users; whilst UNION is more used by the power users. The pattern is quite similar when we compare end-users with developers; the latter make less use of all the features except UNION. However, in this case MINUS is appreciably less used by the developers. Only 12% of the end-users claimed never to use MINUS, compared with 38% of the developers.

[7] This percentage was calculated out of a total of 15; one of the developers did not respond to the question about query types.

Table 6. Query features.

	FILTER	OPTIONAL	BIND	FILTER NOT EXISTS	UNION	MINUS
Survey responses (N = 51)						
Index, s.d.	0.90,0.2	0.86,0.3	0.76,0.3	0.71,0.3	0.61,0.3	0.51,0.4
Query log analysis from Gallego et al. (2011) - %age of queries						
DBpedia	49.19%	16.61%			11.84%	
SWDF	47.28%	0.41%			0.46%	
Query log analysis from Rietveld and Hoekstra (2014) - %age of queries						
DBpedia	15.11%	3.14%			4.46%	
YASGUI	30.35%	1.71%			24.04%	

Property paths

Of the 52 respondents who replied to the question whether they used property paths, 37 (71%) answered that they did. There was no appreciable difference between the light users and the power users nor between end-users and developers. Of the 37 who use property paths, 34 answered each of a subsequent ten questions about frequency of use of the syntax forms described in the SPARQL1.1 standard[8]. The responses to these questions are shown in Table 7. Note that four of the syntax forms are little used: brackets to deviate from the natural order of precedence as specified in the standard; and the three forms of negated predicate sets.

Table 7. Property path syntax forms; survey responses (N = 34)

Syntax form	Index, s.d.	Syntax form	Index, s.d.
Sequence path: elt1/elt2	0.80,0.3	Negated predicate set with forward predicates: !iri or !(iri_1| … | iri_n)	0.16,0.3
Zero or more uses of elt: elt*	0.80,0.3		
One or more uses of elt: elt+	0.66,0.3	Negated predicate set with reverse predicates: !^iri or !(^iri_1| … |^iri_n)	0.07,0.2
Alternative paths: elt1|elt2	0.56,0.3		
Inverse path: ^elt	0.40,0.4	Negated predicate set with forward and reverse predicates: !(iri_1| … | iri_l|iri_m| … |iri_n)	0.07,0.2
Zero or one uses of elt: elt?	0.37,0.4		
Brackets used for precedence	0.18,0.3		

The Gallego et al. (2011) study did not consider property paths, having been undertaken before their introduction in SPARQL 1.1. The Rietveld and Hoekstra (2014) study did not consider the individual property path constructs but did include the four patterns using property paths[9] in their analysis. It is for this reason that their data in Table 3 do not sum to 100%. They found that 6.19% of the DBpedia triple patterns used a property path, whilst for the YASGUI data this figure was 2.94%. In both

[8] https://www.w3.org/TR/sparql11-query/; see section 9.1 of the standard.

[9] I.e., the four possibilities generated by beginning the path with either a variable or a constant (URI or literal) and terminating with a variable or constant: V*C, V*V, C*V, and C*C.

cases, the dominant property path pattern was V*C (i.e. beginning with a variable and terminating with a URI or literal). Bonifati et al. (2017) provide information on property path usage and also indicate very limited use of negated property paths.

A number of syntax forms were originally proposed which were not included in the final SPARQL1.1 recommendation[10]. Respondents who use property paths were asked about their usage of these forms, were they to be available in the SPARQL standard. Of the 37 who previously indicated that they used property paths, 30 responded to this set of questions. The results are shown in Table 8. Whilst care is always required in interpreting the answers to hypothetical questions, the data does suggest some support for these features. Indeed all the features in Table 8 are appreciably more popular than the four least popular syntax forms of Table 7.

Table 8. Property path syntax forms not in SPARQL1.1; survey responses (N = 30)

Syntax form	**Index, s.d.**	**Syntax form**	**Index, s.d.**
Fixed number of occurrences of a subpath	0.45,0.4	Between m and n occurrences of a subpath	0.38,0.3
Between 0 and m occurrences of a subpath	0.45,0.3	n or more occurrences of a subpath	0.37,0.3

A final question about property paths asked whether respondents were likely to use complex property paths requiring nesting or recursion. To illustrate, two examples were given and represented diagrammatically, as shown in Figures 2 and 3. Both examples were taken from the online appendix to Angles et al. (2017). The query represented by Figure 2 is for pairs of individuals who have co-starred together, or are linked by a train of co-stars (i.e. forming transitive closure of 'co-star'), but such that at least one of the pair, and all intermediate individuals have also directed a movie. This query can be achieved using a nested regular expression. The query represented by Figure 3 is for pairs of individuals who have co-starred in a movie and co-directed a (possibly different) movie, or linked by a train of such co-stars / co-directors. This query can be achieved using recursion, e.g. as with a Datalog program.

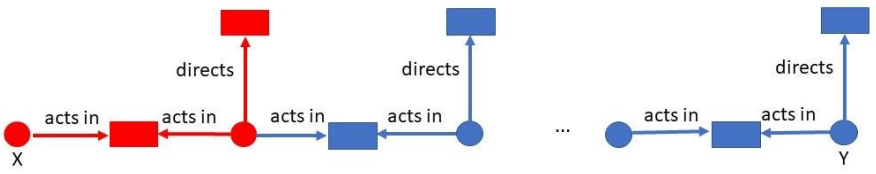

Fig. 2. Query pattern formed with a nested regular expression; the query represents the transitive closure of the subpattern in red (on far left).

[10] See https://www.w3.org/TR/sparql11-property-paths/

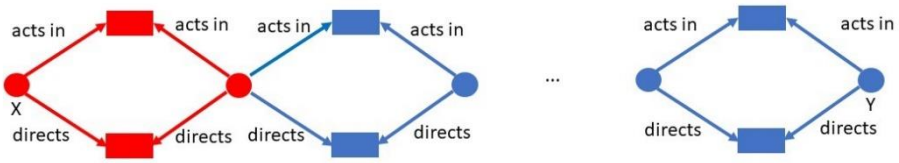

Fig. 3. Query pattern formed using recursion; the query represents the recursive application of the subpattern in red (on far left).

Of the 47 who responded to this question, the majority (70%) were in the 'occasionally' category. 13% responded with 'frequently' and 17% with 'never'. The index, calculated on the same basis as for previous questions, was 0.48. This suggests that, were such syntactic forms to be available, they would enjoy some usage.

5 Difficulties

There were three questions which related to the difficulties people experience in creating queries. The questions were similar, but had slightly different emphases.

Difficulties preventing query creation

Respondents were asked whether there were any queries which they would like to use, but do not, for particular reasons. Table 9 shows the responses to this question.

Table 9. Difficulties preventing query creation - %age of responses

Difficulty	**%age**	**N**	**Difficulty**	**%age**	**N**
Hard to write or understand	26%	42	Not available	30%	37
Speed or other aspect of performance not acceptable	56%	39	Other	7%	29

Associated with the first of these response options, a number of respondents commented on the complexity of queries, including nested queries. The second generated a number of comments about timeouts, e.g. the Wikidata Query Service 60 seconds timeout. A respondent commented that "there are probably ways to optimize" but "that gets into the 'hard to write or understand'". Amongst the requirements for additional features, there were four which extended the functionality of property paths, e.g. finding a path between two individuals and returning a traversed path. Another requirement was to use CONSTRUCT to create a temporary graph over which queries could be run. This approach has, in fact, been suggested by Reutter et al. (2015). A few comments were concerned with grouping and aggregation. Finally, associated with the 'other' category, there was a request for recursive queries.

Difficulties with SPARQL

Respondents were asked to describe any particular difficulties they had with SPARQL. This elicited 28 responses. Eight respondents' comments related to query

efficiency, e.g. the occurrence of timeouts; four of these described difficulties in understanding or predicting what causes inefficiency. Three respondents had concerns relating to documentation. Three were concerned about variability, e.g. between SPARQL endpoints. Three mentioned the need for additional features. One of these cited the need to return paths from a graph, and the need for recursion; the other two were more detailed additional language features. Three referred to complexity or difficulties of understanding, although one of these suggested that "the problem is usually not SPARQL but the source model". Two referred to SPARQL endpoints; one to the difficulty of maintaining resilient and responsive endpoints, the other concerned about their availability. One thought procedurally and therefore had difficulties with SPARQL's declarative approach; the respondent wanted "an alternative syntax that better mirrors the underlying structure of the query".

Query inhibitors

The final question in this category asked if respondents were prevented from creating the kind of queries they wanted, either by the limitations of the tools available, by the difficulty of conceiving queries, or for some other reason. Table 10 shows the percentage experiencing difficulties in each of the non-exclusive categories.

Table 10. Difficulties inhibiting query creation - %age of responses

Inhibitor	**%age**	**N**
Limitations of tools available	49%	39
Difficulties of conceiving queries	32%	41
Other	20%	30

There were 14 comments associated with the limitations of tools. Five of these were concerned with execution time, four complaining about timeouts, and another commenting that triplestore query engines are sometimes not fast enough. The remainder covered a disparate range of topics, many specific to implementations. The most general comments referred to the difficulty of debugging SPARQL queries and to the need for property paths. Regarding the former, the respondent sometimes had "to remove elements and gradually add back pieces". Regarding the latter, the respondent wanted "to run queries about chains of indefinite length with particular conditions at each link".

There were eight comments associated with the second category of response. One respondent commented on the difficulty of complex property paths and wanted "a good graph visualisation tool highlighting specified paths in real-time". Another commented that some queries required much testing and re-reading of the SPARQL manual. A third respondent admitted to sometimes abandoning a complex query and creating "multiple simple queries instead". The remaining comments were not related to the conception of queries; three were concerned with aspects of documentation.

Finally, there were seven comments associated with the 'other' category. One comment referred to the problem of timeouts, which made it necessary to reduce the query or "slice the data". One respondent noted the "lack of recursion, iteration, and

results formatting" and another wanted "better standard function libraries". Another commented that the limitations of SPARQL were not intuitive; so that this led to a realisation "oh, I cannot do what I want".

6 Query construction

There were three questions which asked about the process of query construction, with different emphases and response options.

Approach to query design

Respondents were asked which of four non-exclusive options, plus 'other', described their approach to designing a query; with the choice of specifying 'frequently', 'occasionally' or 'other'. Table 11 shows the responses to these options. The use of previous queries was universal[11]; one respondent commented on having "quite a large collection". Similarly, all but one respondent sometimes worked directly in the query language. On the other hand, a first impression is that respondents made little use of their own representations. However, closer examination of the data revealed that 38% of the 53 respondents made some use of either an internal or an external representation, or both. Comments included a number of references to graphical representations, one use of pseudo-code, and use of Anzo[12], which generates queries from a model. Associated with the 'other' response were four references to use of Wikidata facilities, e.g. the query builder and query helper, plus a reference to Anzo.

Table 11. Approach to designing a query

Approach	Index, s.d.	N	Approach	Index, s.d.	N
Refer to a previous similar query	0.81,0.2	51	Use own internal representation	0.21,0.4	48
Work directly in query language	0.86,0.2	52	Use own external representation	0.18,0.3	49
			Other	0.13,0.3	38

Mental processes

Respondents were asked about their mental processes when creating a model. Table 12 shows the responses to the non-exclusive options. The motivation for this question is the view that some people prefer to reason graphically, and others verbally (Ford, 1995). The majority (53%) of respondents to the question used both forms of reasoning. An appreciable number (41%) never used a graphical approach, whereas only a small number (14%) never used a verbal approach; 7% claimed not to use either approach. One respondent admitted to having a desk "messy with circle-and-arrow sketches". Another respondent thought in sentences "subject predicate object".

[11] I.e. all responses were either 'frequently' or 'occasionally'.

[12] From Cambridge Semantics: https://www.cambridgesemantics.com/

There was also a respondent who, "on rare occasions … used features such as Jena's[13] ability to expose the algebra".

Table 12. Mental processes when designing a query

Mental process	Index, s.d.	N	Mental process	Index, s.d.	N
Graphical, e.g. visualization	0.41,0.4	51	Words and symbols	0.74,0.4	51

Procedural versus declarative approaches to query construction

The final question in this category asked whether, when creating a query, respondents thought in terms of how the solution is arrived at (procedural) or what the solution should look like (declarative). Table 13 shows the two non-exclusive responses. Most people use both approaches. Two respondents commented on the need to think procedurally to create efficient queries. On the other hand, one respondent left the creation of an efficient query to the optimizer. Another respondent resorted to procedural thinking when a timeout occurred.

Table 13. Procedural and declarative approaches to query design

Approach	Index, s.d.	N	Approach	Index, s.d.	N
How solution arrived at	0.66,0.3	47	What solution should look like	0.77,0.3	49

7 General questions

There were two questions which were very general, one about additional features users would like to see, and another giving an opportunity to make final comments.

Additional features

This question asked explicitly what additional features or differences of approach, e.g. alternative syntaxes, respondents would like to see introduced into SPARQL. The question elicited a wide range of responses, a few of which were repetitions of points made earlier.

Six respondents raised issues which related to standardization, in various guises. One respondent wanted to see the Blazegraph[14] extensions incorporated into the SPARQL standard. Other suggestions included the definition of simple subsets of SPARQL; more standardized functions; and standardized support for text indexes.

Five respondents referred, directly or indirectly, to property path features. One of these confirmed a requirement for 'between m and n occurrences of a subpath', as discussed in Section 4. Another wanted path finding between two resources, and another to count the edges between subject and object.

Concerns about efficiency were also present. One respondent noted that query optimizers usually did not interpret the query but rather an underlying language, and that

[13] Jena is an open-source framework providing Semantic Web tools, see https://jena.apache.org/
[14] Blazegraph's uses include the Wikidata Query Service: https://www.blazegraph.com.

this affected the efficiency of filters. Another wanted "benchmarks about the expensiveness of a query"; although admitting that this might be more an issue for the endpoint software than for SPARQL itself. A third suggested "control structures for conditional execution to avoid timeouts".

Other features sought included recursion, which was mentioned by three respondents, and aggregation, mentioned by two; one of these wanted more aggregation operators, the other wanted the ability to order by arbitrary expressions. One respondent reiterated a requirement, previously noted in Section 5, to use CONSTRUCT to create temporary graphs. Another respondent found CONSTRUCT "painful and verbose and hard to post process client side", comparing SPARQL unfavourably with GraphQL[15]. Related to this, although not mentioning CONSTRUCT explicitly, another respondent wanted 'to randomly subsample a set of results before passing it on to the next step in the query'. Other requirements included: better search facilities, based on keyword search, the use of a distance metric, and approximate results; improvements to federated queries, e.g. resilience to component failure and transparency in authentication; and an improved and standardised feature for obtaining the full context of a node.

Final comments

16 respondents took the opportunity to make final comments. Eight were in praise of the language. Of the more critical comments, there was one reference to the timeout problem; the respondent wanted a warning when a complex query had been created. Another was concerned about the cumbersome syntax, making "a scratch pad of copy-paste snippets … essential". One respondent raised the issues of infrastructural performance, stability and resiliency; this respondent suggested the use of Linked Data Fragments and hybrid approaches involving SPARQL.

8 Recommendations

In this section we make some recommendations to those developing tools and to those concerned with the further development of the SPARQL standard.

Recommendations for tool developers

Section 4 suggested a division into *light users* and *power users*. Although, in reality, these are extreme points of a range, it would be useful for tool developers to bear in mind the different needs of these users.

The survey identified a concern with timeouts. It was not that respondents disputed the need for the latter, rather that they wanted to better understand when their queries would be computationally expensive. It would be useful to warn users when a query is likely to be computationally expensive. This certainly does not need to be precise; a statistical approach which gave a reasonable indication in the great majority of cases would be valuable. Additionally, users need a simple, clear model of query

[15] GraphQL allows the structure of the required data to be defined: https://graphql.org/

execution to help them assess the likely cost of a query. Again, this does not have to be precise; it would be sufficient to have an approximation which, on most occasions, gives users a rough idea of the cost of their query.

Also relevant here, there were suggestions about extending the search paradigm, e.g. through standardized text indexes, distance metrics, and approximation.

Recommendations for those developing the SPARQL standard

Users made a wide range of suggestions for enhancements to the language. Many of these related to property paths, where some users are looking for more sophisticated features, e.g. finding the path between two resources, returning the path length, and creating path queries with particular conditions at each link. More generally, some users wanted the inclusion of recursion. Other comments related to aggregation, e.g. more aggregation operators and the ability to order by arbitrary expressions.

Finally, related to the subject of timeouts considered previously, it would be useful to define subsets of the language e.g. for particular kinds of applications, such that when users stayed within those subsets they could be reasonably confident of avoiding very expensive queries. This would be analogous to the profiles of OWL.

References

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2017). Foundations of Modern Query Languages for Graph Databases1. *ACM Computing Surveys (CSUR)*, *50*(5), 68.

Bielefeldt, A., Gonsior, J., & Krötzsch, M. (2018). Practical Linked Data Access via SPARQL: The Case of Wikidata. In *Proc. WWW2018 Workshop on Linked Data on the Web (LDOW-18). CEUR Workshop Proceedings, CEUR-WS. org*.

Bonifati, A., Martens, W., & Timm, T. (2017). An analytical study of large SPARQL query logs. *Proceedings of the VLDB Endowment*, *11*(2), 149–161.

Buil-Aranda, C., Ugarte, M., Arenas, M., & Dumontier, M. (2015). A preliminary investigation into SPARQL query complexity and federation in Bio2RDF. In *Alberto Mendelzon International Workshop on Foundations of Data Management* (p. 196).

Ford, M. (1995). Two modes of mental representation and problem solution in syllogistic reasoning. *Cognition*, *54*(1), 1–71.

Gallego, M. A., Fernández, J. D., Martínez-Prieto, M. A., & de la Fuente, P. (2011). An empirical study of real-world SPARQL queries. In *1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011) at the 20th International World Wide Web Conference (WWW 2011), Hyderabad, India*.

R Core Team. (2014). *R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. 2013*. ISBN 3-900051-07-0.

Raghuveer, A. (2012). Characterizing machine agent behavior through SPARQL query mining. In *Proceedings of the International Workshop on Usage Analysis and the Web of Data, Lyon, France*.

Reutter, J. L., Soto, A., & Vrgoč, D. (2015). Recursion in SPARQL. In *International Semantic Web Conference* (pp. 19–35). Springer.

Rietveld, L., & Hoekstra, R. (2014). Man vs. machine: Differences in SPARQL queries. In *Proceedings of the 4th USEWOD Workshop on Usage Analysis and the Web of of Data, ESWC*.