Navigating the Maze of Wikidata Query Logs

Angela Bonifati
Lyon 1 University
angela.bonifati@univ-lyon1.fr

Wim Martens
University of Bayreuth
wim.martens@uni-bayreuth.de

Thomas Timm
University of Bayreuth
thomas.timm@uni-bayreuth.de

ABSTRACT

This paper provides an in-depth and diversified analysis of the Wikidata query logs, recently made publicly available. Although the usage of Wikidata queries has been the object of recent studies, our analysis of the query traffic reveals interesting and unforeseen findings concerning the usage, types of recursion, and the shape classification of complex recursive queries. Wikidata specific features combined with recursion let us identify a significant subset of the entire corpus that can be used by the community for further assessment. We considered and analyzed the queries across many different dimensions, such as the robotic and organic queries, the presence/absence of constants along with the correctly executed and timed out queries. A further investigation that we pursue in this paper is to find, given a query, a number of queries structurally similar to the given query. We provide a thorough characterization of the queries in terms of their expressive power, their topological structure and shape, along with a deeper understanding of the usage of recursion in these logs. We make the code for the analysis available as open source.

CCS CONCEPTS

• **Information systems** → **Query log analysis**; *Query languages for non-relational engines*; *World Wide Web*; *Information retrieval query processing*; • **Theory of computation** → *Database query languages (principles)*;

KEYWORDS

Query Log Analysis; Query Shapes; Query Similarity Search; SPARQL endpoint; Knowledge Graph

ACM Reference Format:
Angela Bonifati, Wim Martens, and Thomas Timm. 2019. Navigating the Maze of Wikidata Query Logs. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313472

1 INTRODUCTION

Wikidata [31] is a free collaborative knowledge base that has been characterized by a gigantic growth in terms of number of edits, number of users and developers, and amount of automated software since its inception in 2012 by the Wikimedia Foundation. The interplay between user and bot activities on Wikidata is for instance an interesting subject to study in order to make sense of the quality of the newly added items produced by the massive numbers of edits in the knowledge base [27]. Contrarily to Wikidata data dumps, which are readily available and allow a flurry of analyses, the activity of both humans and bots on the Wikidata SPARQL endpoints can only be investigated since recently, thanks to the release of large anonymized query logs.[1] These query logs represent a rich swath of information about the robotic and organic query traffic on Wikidata and deserve our attention for further investigation, in particular to understand the structure of complex queries. A preliminary analysis of the Wikidata query logs bootstrapped with a recent paper by Malyshev et al. [20], who first introduced the Wikidata SPARQL service WDQS[2] and pinpointed its technical characteristics and current usage. They also provided a classification of the Wikidata queries into robotic and organic requests that we readily adopt in the present paper and that we recapitulate in Section 9. They made several observations on which we build in this paper, namely that robotic query traffic dominates organic query traffic in terms of volume and query load, and that robotic queries are issued by a single source whereas organic queries are typically multi-source. They also identified a massive presence of recursive queries in these logs, whose prominent fragment consists of queries only containing joins and property paths,[3] also known as conjunctive 2-way regular path queries (C2RPQs) in the literature. These massive logs of recursive queries are the first encountered so far, as opposed to negligible percentages of these queries in previous large-scale logs, including DBpedia queries [9].

C2RPQs are the basic building blocks of graph query languages in the literature of RDF and graph databases. They allow to express navigational patterns on the graph instances by leveraging regular expressions, also known as Property Paths in the SPARQL 1.1 specification [15]. In Wikidata, they are particularly important since they emulate ontological reasoning in SPARQL and also express complex label-constrained reachability queries.

We focus our analysis on these recursive queries in the Wikidata logs and further extend the class of C2RPQs by incorporating the use of Service, Values, Bind, Filter, and Optional. Indeed, Service, Values, and Bind occurred only rarely in other massive logs [9] but are very prominent here. This fragment, which we call C2RPQ+, constitutes more than 85% of the valid queries in the logs.[4] The consideration of this fragment leads us to redesign many tests conducted in [9], among which the triple count and shape analysis, which were based on the less expressive *conjunctive queries*.

Although the distinction between robotic and organic query traffic has been introduced in [20], it has never been used in the

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313472

[1] Accompanying the publication by Malyshev et al. [20], the Knowledge-Based Systems Group at TU Dresden released anonymised logs of several hundred million SPARQL queries from the Wikidata SPARQL endpoint [6].
[2] Whose user interface can be found at https://query.wikidata.org/, while the raw SPARQL endpoint is at https://query.wikidata.org/sparql
[3] After removal of subqueries that correspond to Wikidata's labeling service. They also allow the use of Values, which indeed can be done to some extent, see Section 6.
[4] In fact, the 85% only refers to a subfragment that is *suitable for graph shape analysis*.

complex analysis that we carry out in this paper regarding the types of property paths, the computation of triples when property paths are present and the shape analysis of the C2RPQ+ fragment. We also provide a view from rather different angles of these logs by considering them with or without duplicates and by separating the analysis of successfully executed and timeout queries, the latter being analyzed for the first time in our study. We also developed a query similarity search tool capable of identifying from an initial query the set of structurally similar queries by using tree-edit distance. This tool allows to further inspect the logs by having a specific query in mind and in a sense permits to reproduce and reapply the previous complex analysis to the obtained sets for future studies.

We focus in this study on the following research questions: What is the distribution of query sizes? Which qualifiers are popular in queries? How are property paths used and what is their structure? How prominent are conjunctive queries (and variants thereof)? What is the shape and (hyper)treewidth of queries? Given a query, can we find in a subset of the logs the queries that are structurally similar to it? Furthermore, we are also interested in meta-questions, such as: Are there differences between robotic and organic queries? Does a comparison with an earlier study [9] allow us to identify some trends?

As a general remark, our study is significantly more extensive than what we can present in this paper. Our code for the analysis is publicly available [11].

2 DATA SETS

Our corpus consists of all queries in the Wikidata query logs that were recently made publicly available [6]. Precisely, the queries considered in this paper have been downloaded on October 12th, 2018. These logs are anonymized and represent queries that were submitted to the Wikidata SPARQL endpoint from June 12th until September 3rd in 2017. The same queries have been considered in the work of Malyshev et al. [20], as discussed in Section 1. We have partitioned these log files in four disjoint sets: queries for which the HTTP request was successful[5], further partitioned in organic (OrganicOK) and robotic queries (RoboticOK); and timeout queries, further partitioned in organic (OrganicTO) and robotic queries (RoboticTO).

Each query in the downloadable log files has an annotation that indicates if it was classified as a bot or user query by Malyshev et al. [20]. We used the same classification.[6] Sometimes we use OK, (resp., TO) to refer to OrganicOK ∪ RoboticOK (resp., OrganicTO ∪ RoboticTO) for brevity. The TO queries have not been considered in the work of Malyshev et al. [20].

Table 1 describes, for each of the log types, its number of queries (Total #Q), number of valid queries, i.e., queries that parse using Apache Jena 3.7.0 (Valid #Q), and the number of valid queries after removal of duplicates (Unique #Q). For duplicate removal, we considered two queries to be the same if they are the same string after whitespace removal.

[5]HTTP code 200.

[6]Our number of queries in Organic is slightly higher than the number of queries reported on the download page of the query logs. We believe that the Dresden file may be incomplete, since we found organic queries in the "all queries" log files that do not show up in the organic subset in [6].

Table 1: The query logs in our corpus

Source	*Total #Q*	*Valid #Q*	*Unique #Q*
RoboticOK	207,505,296	207,464,954	34,523,883
OrganicOK	661,769	651,385	251,994
RoboticTO	33,616	33,465	3,168
OrganicTO	14,528	14,087	8,729
Robotic	207,538,912	207,498,419	34,527,051
Organic	676,297	665,472	260,723
OK	208,167,065	208,116,339	34,775,877
TO	48,144	47,552	11,897
Total	208,215,209	208,163,891	34,787,774

Table 2: Distribution of Select, Ask, Construct, and Describe

		Valid #Q	*Valid %*	*Unique #Q*	*Unique %*
Select	Robotic	206,006,783	99.28%	34,261,882	99.23%
	Organic	664,323	99.83%	260,114	99.77%
Ask	Robotic	1,127,396	0.54%	73,019	0.21%
	Organic	306	0.05%	132	0.05%
Construct	Robotic	188,088	0.09%	24,587	0.07%
	Organic	516	0.08%	217	0.08%
Describe	Robotic	176,152	0.08%	167,563	0.49%
	Organic	327	0.05%	260	0.10%

Throughout the entire paper, we will use the following notation to discuss data sets. Whenever we report a number or a percentage in the format X (Y), the number X refers to the Valid and the number Y refers to the Unique sets of queries. This notation allows the reader to stay informed throughout the paper about the queries that the endpoint actually receives (Valid) and about those without duplicates in this set (Unique).

Already from Table 1 we can make a number of interesting observations. One simple observation is that the robotic logs contain many more duplicates than the organic logs. Indeed, whereas Organic contains 39.18% unique queries, Robotic only contains 16.64% unique queries. A second observation is that, even though queries do not timeout very often, organic queries time out 100 times more often than robotic queries. The fraction of OrganicTO queries to Organic queries is 2.12% (3.35%), whereas the fraction of RoboticTO to Robotic queries is 0.02% (0.01%). In the set of *unique* timeout queries, a whopping 73.37% are organic.

Data Sets for Analysis. We noticed that almost all Describe queries do not have a body. For this reason, and because Describe queries do not have a well-defined semantics, we solely focus from now on in the paper on the Select, Ask, and Construct queries. This subset of the corpus has 207,987,412 valid queries, 34,619,951 of which are unique. The robotic subset contains 207,322,267 (34,359,488) and the organic subset contains 665,145 (260,463) queries. In all cases, these sets encompass more than 99.5% of the total sets in Table 1, see Table 2.

3 BASIC DEFINITIONS

We assume familiarity with SPARQL, but recall the very basics of the language. The presentation of the following definitions is strongly based on Picalausa and Vansummeren [26] and Bonifati et al. [9]. A *SPARQL query Q* can be seen as a tuple of the form

(*query-type*, *pattern P*, *solution-modifier*).

Here, *query-type* is one of Select, Ask, Construct, and Describe. The *pattern* is the central component of the query, which we will discuss in more detail next, and the *solution-modifier* is for performing aggregation, grouping, sorting, duplicate removal, and returning only a specific window (e.g., the first ten) of the solutions returned by the pattern.

Patterns. By $\mathcal{I}$, $\mathcal{B}$, and $\mathcal{L}$ we denote the sets of *IRIs*, *blank nodes*, and *literals* from SPARQL, respectively. Let $\mathcal{V} = \{?x, ?y, ?z, ?x_1, \ldots\}$ be an infinite set of variables, disjoint from $\mathcal{I}$, $\mathcal{B}$, and $\mathcal{L}$. As in SPARQL, we always prefix variables by a question mark. A *triple pattern* is an element of $(\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$. A *property path* is a regular expression over the alphabet $\mathcal{I}$. A *property path pattern* is an element of $(\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times pp \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$, where pp is a property path. A *SPARQL pattern* P is an expression generated from the following grammar:

$$P ::= \quad t \mid pp \mid Q \mid P_1 \text{ And } P_2 \mid P \text{ Filter } R \mid P_1 \text{ Union } P_2 \mid P_1 \text{ Optional } P_2 \mid \text{Bind } X \text{ AS } v \mid \text{Service } n\ P \mid \text{Values } tup\ T$$

Here, t is a triple pattern, pp is a property path pattern, Q is again a SPARQL query, and R is a so-called *SPARQL filter constraint*. SPARQL filter constraints R are built-in conditions which can have unary predicates, (in)equalities between variables, and Boolean combinations thereof. Bind associates a unary expression to a single variable v. Service calls a remote service with name n and sends it a pattern P. Finally, Values binds a tuple tup to values in a given table T. We note that property paths (pp) and subqueries (Q) in the above grammar are new features since SPARQL 1.1. We refer to the SPARQL 1.1 recommendation [15] and the literature [25] for the precise syntax of filter constraints and the semantics of SPARQL queries. We write Vars(P) to denote the set of variables occurring in P.

We illustrate by example how parts of our definition correspond to real SPARQL queries. The following query comes from Wikidata ("Locations of archaeological sites", from [32]).

```
SELECT ?label ?coord ?subj
WHERE
{ ?subj wdt:P31/wdt:P279* wd:Q839954 .
  ?subj wdt:P625 ?coord .
  ?subj rdfs:label ?label FILTER(lang(?label)="en") }
```

The query uses the property path `wdt:P31/wdt:P279*`, the literal `wd:Q839954`, and the triple pattern `?subj wdt:P625 ?coord`. It also uses a filter constraint. In SPARQL, the And operator is denoted by a dot (and is sometimes implicit in alternative, even more succinct syntax).

4 QUERY SIZES AND OTHER COUNTING MEASURES

We report in Figure 1 the distribution of length of the queries in terms of the number of their triples. Contrarily to other triple count distributions reported in previous studies, we include property paths in our metric. The reason is that a property path pattern of the form `?x wdt:P31/wdt:P279 ?y` is a shorthand notation for *two* triples and we feel that it should be considered as such. We therefore count property path patterns as follows. Let pp be a property path encoded as a regular expression. We say that the *size* of pp is the number of alphabet symbols in the regular expression. For example, the property paths wdt:P279/wdt:P279 and wdt:P279/wdt:P279* have size two. The query illustrated in Section 3 has size four: it contains two triples of size one and one triple of size two.[7]

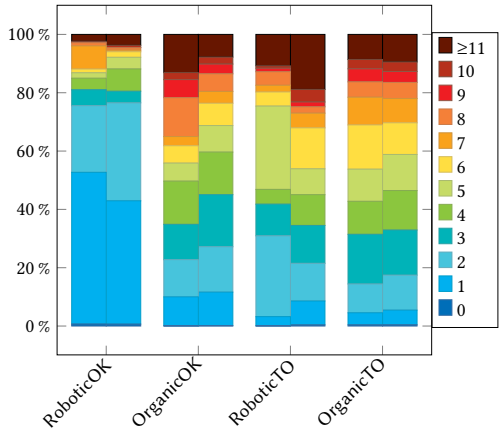

Figure 1: Percentages of queries with corresponding sizes (number of triples plus sizes of property paths) for each dataset — Valid (left) versus Unique (right). The sizes are reported on the right-hand side and range from 0 to ≥ 11.

We analyzed the triples and property paths inside the Select, Ask, and Construct clauses and explicitly exclude the queries with a Describe clause, as discussed in Section 2. The results for the four data sets can be found in Figure 1. Here, each bar is split in two, where the left side represents the Valid and the right side the Unique version of each data set.

The conclusions of this specific analysis are similar to those made in previous work [7, 20], assessing that user queries are more diverse than robotic ones. Nevertheless, the triple count distributions shown in Figure 1 shows the breakdown in terms of the other dimensions considered in our analysis, encompassing the valid and unique queries and the OK and timeout queries. The timeout queries for instance show a relatively higher complexity in terms of number of triples than the OK ones, and this observation also applies to RoboticTO queries, that are thus quite different from RoboticOK queries. This seems to suggest that timeout queries are queries that failed because of a higher number of triples, which could be interesting to consider in graph query evaluation and optimization studies. The information about the average number of triples confirmed this, since Valid queries have on average 2.58 (2.65) triples, whereas timeout queries have 5.65 (5.94) triples. As a side remark, the highest number of triples that we observed is in the RoboticOK Valid logs and is equal to 67, which was found in 68 queries (in 34 queries in the RoboticOK Valid logs, respectively). The largest size of a property path triple was 19.

Number of Constants and Variables. Counting the number of triples is only one possible measure of the complexity of a query. We enrich this analysis by considering further characteristics of the triples, i.e., whether the triples contain variables or constants. This information is useful for the shape analysis that we conduct in Section 6, in which we show actual differentiations in the obtained

[7] Notice that the query size computed as described above even with Kleene-star ('*') and transitive closure operators ('+') does not depend on the length of the actual paths in the graph instance when evaluating the query.

Table 3: Number of constants and variables in the triples (Unique logs)

# Const	# Organic	# Robotic	# Vars	# Organic	# Robotic
0–9	239,003	33,485,122	0–9	255,383	33,654,282
10–19	21,307	480,936	10–19	3,999	423,415
20–32	153	393,430	20–27	1,081	281,791

shapes by removing or including the constants. We only count the number of distinct variables and constants in these experiments. Table 3 reports the numbers of constants and variables by intervals in the Unique OrganicOK and RoboticOK logs.

Precisely, we counted the number of different variables (different constants, respectively) of each query in the logs and reported the total number of queries that have this number. For conciseness, we aggregate these numbers into intervals in Table 3. We can observe that for numbers of constants and variables greater than 11, the queries with these numbers of constants are more abundant than queries with these numbers of variables.

Usage of Wikidata Properties. We then analyzed the organicOK Unique query logs in order to get a feel of the usage of different Wikidata properties (P856, P31,...) appearing in the queries. The total number of properties found in these logs are 881,490. These are divided into two major namespaces at a top level: `www.wikidata.org` (805,196), and `www.w3.org` (70,829). We report in Figure 2 a sunburst diagram showing the segmentation of the Wikidata properties inside the largest top-level namespace qualifier `www.wikidata.org`. In order to avoid clutter and for ease of presentation in the paper, we have solely annotated the sunburst with the properties whose occurrences are above a given threshold (6,000). More views about the sunburst including the other top-level properties and with complete information about all the number of occurrences, is available via an interactive version of the diagram [29]. Hovering over the various segments of the rings provides the information omitted here in order to avoid clutter in Figure 2.

Subqueries and Projection. Roughly 1% of the queries in the Unique Logs use subqueries. This number goes down to 0.37% for the corresponding subqueries in the Valid logs.

We also ran a test for mining the number of queries that use projection. Projection is a cause of complexity increase of query evaluation for CQ queries, that goes from NP-complete is projection is present to PTIME if projection is absent [5, 18]. Similarly to [9] we use the test for projection in Section 18.2.1 in the SPARQL 1.1 recommendation [15]. Out of the valid queries from Table 2, we found 25,569,947 (12.28%) queries that use projection.[8] Out of the valid Organic queries as reported in Table 2, the amount of queries with projection even rises to 28.85%. These percentages become 18.01% and 28.05% for the unique valid and Organic queries, respectively. In particular, they are much higher than the 13.12% Select/Ask queries found within the Unique query logs of the DBpedia corpus in [9].

5 PROPERTY PATH ANALYSIS

Overall, 49,971,258 (13,480,433) queries in our logs use property paths, which amounts to a total of 24.03% (38.94%) of the entire logs. In these queries, we found 165,343 (82,764) property paths in

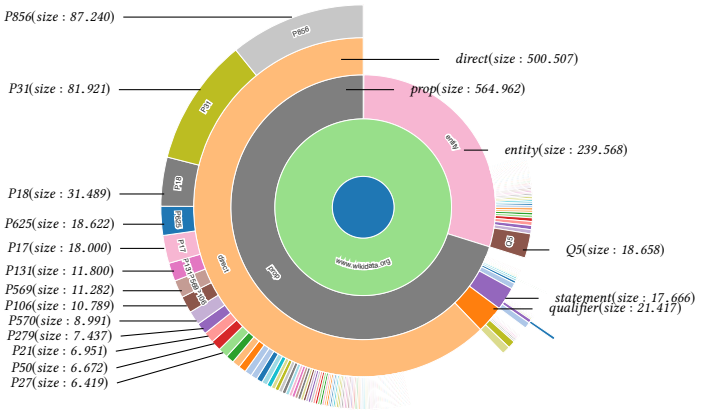

Figure 2: Sunburst distribution of the property qualifiers in Wikidata queries (Unique OrganicOK query logs.). An interactive version is available online at [29].

organic queries and 55,168,101 (14,106,489) in robotic ones. (Notice that the same query can contain multiple property paths.) This massive corpus of property paths is the first encountered so far, and significantly (57x) greater than the one found in a large corpus in [9] featuring only 247,404 property paths.

Both organic and robotic property path corpuses are interesting for an analytical study and deserve a deeper inspection in order to classify the occurring path expressions into distinct types. Indeed, when we looked at the *structure* of these path expressions, we found 234 different types of organic expressions, compared to only 64 types of robotic expressions. Such a thorough classification revealed the different characteristics of the organic property paths with respect to the robotic ones, as the former exhibit more variety and heterogeneity than the latter despite their lower occurrences. Here, the *type* of a property path is obtained as follows. We replace each variable or IRI by letters from the alphabet in increasing order. (If a variable or IRI is repeated in the property path, we replace it by the same alphabet letter.) For example, `wdt:P31*/wdt:P279*` is of the type a^*b^* and `wdt:P31/wdt:P31*/wdt:P279*` is of the type aa^*b^*.

Robotic Property Paths. Table 4 contains a summary of the most common types of property paths in robotic queries. The columns with "V" represent results for the *Valid* queries, and the columns with "U" for the *Unique* queries. For succinctness, we aggregated different types together. For example, we aggregated each type with its reverse type. For instance, the row for ab^* also contains the expressions of the form a^*b. Furthermore, we treated $\hat{}a$ ("follow an a-edge in reverse direction") the same as a single label.[9] Finally, we also grouped disjunctions together, denoted by capital letters. In Table 4, a capital letter A denotes a subexpression that matches a *disjunction of at least two symbols*. Empirically, an A either denotes an expression of the form $!a$, $(a|!a)$, or a disjunction of the form $(a_1|\cdots|a_k)$ with $k > 1$. We divided Table 4 into *transitive expressions* (top) and *non-transitive expressions* (bottom). Transitive expressions are those that match arbitrarily long paths (i.e., they

[8] This is a lower bound, since our test is sound but incomplete.

[9] The operator $\hat{}$ is used in 0.80% (1.10%) of robotic and 2.03% (3.18%) of organic queries.

Table 4: Structure of property paths for all robotic queries

Expression Type	*AbsoluteV*	*RelativeV*	*AbsoluteU*	*RelativeU*
a^*	27,850,487	50.48%	1,392,865	9.87%
ab^*, a^+	9,417,166	17.07%	2,816,134	19.96%
ab^*c^*	823,153	1.49%	67,502	0.48%
A^*	328,895	0.60%	51,860	0.37%
ab^*c	122,286	0.22%	1,680	0.01%
a^*b^*	62,784	0.11%	608	
abc^*	27,287	0.05%	4,083	0.03%
$a?b^*$	15,893	0.03%	11,999	0.09%
A^+	4,674	0.01%	2,043	0.01%
Ab^*	1,562		674	
Other transitive	1,643		161	
$a_1 \cdots a_k$	13,382,005	24.26%	9,368,442	66.41%
A	3,043,725	5.52%	381,434	2.70%
$A?$	31,150	0.06%	296	
$a_1a_2? \cdots a_k?$	25,872	0.05%	5,940	0.04%
$\hat{}a$	21,202	0.04%	471	
$abc?$	7,620	0.01%	8	
Other non-transitive	697		289	
Total	55,168,101	100%	14,106,489	100%

use the operators * or $^+$). The empty cells represent values that round down to 0.00%.

Interestingly, we see significant differences between the numbers of expressions in the valid and in the unique sets. Whereas the type a^* accounts for 50.48% of the expressions in the valid data set, this drops to 9.87% in the unique set. On the other hand, concatenations of symbols (type $a_1 \cdots a_k$) represent 24.26% in the valid queries, but over 66% in the unique queries. To further understand this phenomenon, we focused on the valid RoboticOK and OrganicOK and computed the most popular 20 queries containing a property path with Kleene-star. The top most popular robotic query with property paths (having 281,096 occurrences) belong to the second most occurring type in Table 4 since it contains a single path expression of the kind `wdt:P31/wdt:P279`*. The query is in fact a *conjunctive regular path query.* [10]

Organic Property Paths. Table 5 contains results on the Organic datasets. Since we had 234 different types of expressions, we needed to aggregate more aggressively to make the results presentable. We grouped the types into 41 different categories, from which we omitted some in the table due to space restrictions. The main difference with Table 4 is that we also allow a capital letter to denote a single symbol. So, A can denote expressions equivalent to a, $(a_1|\cdots|a_k)$, $!a$, or $(!a|a)$. The other difference is that we grouped the types $(ab)^*$ and $a(bc)^*$ together in the type $\sim(ab)^*$. These expressions stand out from the rest, since they are the *only* type of transitive expressions we found that put length constraints on arbitrarily long paths. Indeed, all other transitive expressions allow paths of arbitrary length, once the length exceeds a certain value. This is not the case for $(ab)^*$, since it only allows paths of even length.

Here, the percentages in the unique sets are quite similar to those in the valid sets. The Organic queries generally contain more challenging property paths to evaluate. On average, organic property paths are also larger than robotic ones. They contain 2.07 (2.01) literals on average, whereas robotic property paths only contain

[10] By looking at the top most popular organic query (with 1,778 occurrences), we noticed that it belongs to the first most occurring type in Table 5 and exhibits the same single path expression. As a side remark, we can notice that the latter query is fairly more complex and uses Union, Filter, Bind, and Service clauses.

Table 5: Structure of property paths for all organic queries

Expression Type	*AbsoluteV*	*RelativeV*	*AbsoluteU*	*RelativeU*
AB^*	57,913	35.03%	28,034	33.87%
A^*	41,777	25.27%	22,071	26.67%
ABC^*	6,497	3.93%	3,044	3.68%
a^*b^*	3,330	2.01%	849	1.03%
ab^*c^*	2,704	1.64%	1,172	1.42%
$a^*B_1?b_2? \cdots b_k?$	1,789	1.08%	422	0.51%
$ab\|c^*d$	1,514	0.92%	534	0.65%
$a^*\|b^*$	347	0.21%	253	0.31%
$abCD^*$	283	0.17%	219	0.26%
ab^*c	113	0.07%	90	0.11%
$a^*\|B$	102	0.06%	76	0.09%
$\sim(ab)^*$	101	0.06%	82	0.10%
ab^*c^*d	86	0.05%	72	0.09%
$ab^*\|c$	70	0.04%	59	0.07%
$a^*b?c$	56	0.03%	27	0.03%
$a^*b^*c^*$	32	0.02%	28	0.03%
$ab^*\|b^+a^*$	16	0.01%	12	0.01%
ab^+c	13	0.01%	12	0.01%
$ab^*\|cd^*$	13	0.01%	12	0.01%
a^*bc^*	11	0.01%	11	0.01%
Other transitive	22	0.01%	20	0.02%
$A_1a_2 \cdots a_k$	31,032	18.77%	15,754	19.03%
A	13,248	8.01%	7,592	9.17%
$a_1? \cdots a_k?$	1,938	1.17%	1,470	1.78%
$A?$	1,178	0.71%	302	0.36%
$ab_1? \cdots b_k?$	1,117	0.68%	529	0.64%
$ab\|c$	27	0.02%	5	0.01%
Other non-transitive	14	0.01%	13	0.02%
total	165,343	100.00%	82,764	100.00%

1.49 (1.89) literals on average. There were even expression types which occurred more often in the TO (timeout) logs than in the OK logs, such as $\sim(ab)^*$. This is interesting, because such expressions are known to be complex (NP-complete) to evaluate under simple path semantics [3].

Additional Insights on Wikidata Property Paths. We conclude the section with a discussion of the differences between Tables 4 and 5 and the results of the property path analysis by Bonifati et al. [9]. The remarkable difference between the present study and the former (which was done on a corpus mainly consisting of DBPedia queries) is that here, a much larger fraction of the queries use property paths. This is probably due to the peculiar characteristics of the Wikidata data. Property paths are often used in queries performing class navigation in Wikidata and emulating ontological reasoning. Wikidata has relatively long paths in the data that are labeled with the same label (and that are popular to query, e.g., InstanceOf paths), whereas DBpedia has comparably shorter paths and is more flat.

The remarkable similarity, however, is in the *structure* of the property paths. Martens and Trautner [21] defined the class of *simple transitive expressions*, which are syntactically very restricted, but covered over 99% of the property paths in the corpus of Bonifati et al. [9]. In our corpus, 1.61% (0.48%) of the robotic and 3.83% (2.72%) of the organic property paths are not simple transitive expressions. The most significant reason why property paths fall out of this fragment is the use of a^*b^* as a subexpression, whereas simple transitive expressions only use one subexpression with Kleene star.

Furthermore, all property paths except 198 (98) are in C_{tract}, which is a broader class introduced by Bagan et al. [3] and which

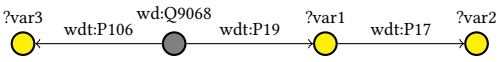

Figure 3: Visualization of a query as a graph

precisely characterizes the set of regular expressions for which the data complexity under simple path semantics is tractable if P ≠ NP.

6 STRUCTURAL ANALYSIS AND RECURSIVE PROPERTIES

Consider the following query, which searches the occupations (P106) of Voltaire (Q9068), and returns the place (P19) and country (P17) of birth.

```
SELECT * WHERE { wd:Q9068 wdt:P19 ?var1 . ?var1 wdt:P17 ?var2
               . wd:Q9068 wdt:P106 ?var3 }
```

This query can be visualized as a graph, see Figure 3. Answering the query essentially boils down to matching this graph in the data and returning answers for each such match.

This query is an example of a *conjunctive query*, i.e., a query that only uses the And operator. For conjunctive queries, there is an extensive body of research that correlates the shape of the graph with the complexity of their evaluation problem (see, e.g., [10, 13, 17, 33]). In particular, *cycles* in the graph play a major role. In general, conjunctive query evaluation is well known to be NP-complete. On an intuitive level, the reason is that, if a conjunctive query has the shape of a k-clique (which is heavily cyclic), deciding if it returns a non-empty result is equivalent to deciding if the data has a k-clique, which is NP-complete. On the other hand, using Yannakakis' algorithm, we can evaluate *acyclic* conjunctive queries in polynomial time [33].

It is known that even a single cycle can have a significant impact on the run-time of queries [9, 16]. Therefore, for queries that behave similar to conjunctive queries, the shape of their graph is important for understanding their complexity. Furthermore, it gives useful insights on the structure of real queries.

In the following, we will define the *graph shape* of several kinds of queries. We focus exclusively on queries that behave similar to conjunctive queries and for which the cyclicity of the graph shape correlates with the complexity of the evaluation. We start by considering conjunctive queries and add other operators one by one.

Bonifati et al. [9] performed a graph shape analysis on the queries in their corpus, but the corpus does not provide much insight for Wikidata, because it only included 308 Wikidata queries. Furthermore, they only considered shapes of conjunctive queries, extended with (safe use of) Filter and Optional.[11] Since only 25.89% (42.48%) of our robotic and 17.89% (21.15%) of our organic queries are eligible for such an analysis, we extended the analysis to incorporate (the safe use of) four extra features: property paths, Bind, Values, and Service. By doing so, we were able to make the analysis applicable to *2 to 3 times more queries in every fragment*, that is, 85.22% (88.39%) of the robotic and 51.50% (65.22%) of the organic queries. Notice that the queries thereby obtained are not merely subsets of the queries that use a given set of keywords. This obviously implies that the shape test is more sophisticated than the keyword test. We explain how we deal with each keyword separately in Sections 6.2–6.4.

[11] By *safe use* we refer to the use of these operators so that the queries still behave similar to conjunctive queries.

6.1 Graph Patterns and Canonical Graphs

In the following we want to define the *graphs* of different types of queries. These graphs will be undirected, that is, $G = (V, E)$ where V is its (finite) set of nodes and E is its set of edges, where an edge e is a set of one or two nodes, i.e., $e \subseteq V$ and $|e| = 1$ or $|e| = 2$.

We need to determine which queries can be adequately represented as graphs. Since graphs are node pairs and SPARQL queries have triple patterns, we need to be careful with the use of variables in the predicate position. In our corpus, 91.68% (96.43%) of the queries only use triple patterns (s, p, o) where p is an IRI, so these queries can be adequately represented as graphs. We also allow $p \in \text{Vars}$ if p is not used elsewhere in the query (in this case, p serves as a wildcard, possibly binding to a value that is returned to the output.). We call such patterns *graph patterns*.

The *canonical graph* of a graph pattern P is the graph $G_P = (V_P, E_P)$ with $E_P = \{\{x, y\} \mid (x, \ell, y)$ is a triple pattern in P and $\ell \in \mathcal{I} \cup \mathcal{V}\}$ and $V_P = \{x \mid (x, \ell, y) \in E_P$ or $(y, \ell, x) \in E_P\}$.

6.2 Conjunctive Queries

Conjunctive queries are the basic building blocks of our shape analysis. In the context of SPARQL, we define them as follows.

Definition 6.1. A *conjunctive query (CQ)* is a SPARQL pattern that only uses the triple patterns and the operator And.

In our corpus, 20.88% (33.69%) of the queries are CQs, which is quite low compared to the study of Bonifati et al. [9]. Part of the reason is that Wikidata queries extensively use Service, most commonly for Wikidata's labeling service [7]. In our logs, 8.38% (12.94%) of the queries use Service in some way.

The good news is that 99.97% (99.997%) of the CQs are graph patterns. We call such CQs *eligible* for graph shape analysis. We note that it is also possible to investigate the shape (or cyclicity) of non-eligible CQs, but we need to consider their *hypergraphs*, see Section 7.

The importance of the shape of conjunctive queries becomes clear in the following result, linking the *treewidth (tw)* of the query's graph to the complexity of query evaluation. The precise definition of treewidth is not important for the paper but, intuitively, treewidth measures *how close the graph is to a tree*. For instance, a tree has treewidth 1 and a k-clique (which is very cyclic) has treewidth k. Queries with tw = 1 are also called *acyclic*.

Theorem 6.2 (cfr. [10, 13, 17]). *Let G be a graph and Q an eligible conjunctive query for which the canonical graph has treewith k. Then it can be tested in time $|G|^{O(k)}|Q|^{O(k)}$ if Q returns a non-empty result on G.*

6.3 Adding Filter and Optional

Bonifati et al. [9] investigated the structure of graph patterns that only use the operators And, Optional, and Filter. In our corpus, 27.72% (44.24%) of the queries are in this fragment. Our extension of CQs with Filter and Optional is very similar to the one of Bonifati et al., that is, we focus on *well-designed* patterns (cf. [25]) with

interface width 1 (cf. [4]). Our main difference is that we now allow binary filter constraints to be edges in the graph.

6.4 Adding Recursion, Bind, Service, and Values

Although the previous tests already classify 53,804,198 (14,649,616) queries to be eligible for graph shape analysis, this is still only 25.87% (42.32%) of our queries. We now discuss how property paths (recursion), Bind, Service, and Values can be incorporated to increase the number of suitable queries to 177,022,071 (30,540,864), or 85.11% (88.22%) of the logs.

Property paths are unproblematic. Conjunctive queries extended with property paths closely correspond to the well known *conjunctive two-way regular path queries (C2RPQs)*, which form a basis of navigational query languages for graphs. Indeed, property paths are very closely related to regular path queries and, due to the ˆ-operator, they can navigate edges in both forward and backward direction, which makes them *two-way*. Although there are some semantical differences between two-way regular path queries and property paths [1, 15, 19], these differences are not crucial for the present analysis.

Definition 6.3. A *conjunctive two-way regular path query (C2RPQ)* is a SPARQL pattern that only uses triple patterns, the operator And, and property paths.

Every C2RPQ that is a graph pattern is suitable. The *graph of a C2RPQ* P is obtained from G_P by adding the edges $\{\{x, y\} \mid (x, pp, y)$ is a property path pattern in $P\}$ to E_P (and adding nodes to V_P if necessary).

Adding Filter and Optional to C2RPQs is analogous to adding them to CQs. The resulting classes of queries are referred to as C2RPQ_F and C2RPQ_OF. As such, we obtain that 46.03% (76.44%) of our queries are C2RPQ_OF queries that are suitable for graph shape analysis.

Concerning Bind, we follow a similar approach to Filter. We call a Bind-condition k-ary if it involves k variables. Unary and binary Bind-conditions are considered to be suitable for graph analysis and are materialized as edges in the graph. Higher-arity Bind-conditions are considered in Section 7.

Values-constructs are essentially used to test if a tuple of variables is in a given set. For instance, the subquery

```
VALUES (?book ?title)
{ (:book1 "Robin Hood")
  (:book2 "Little Red Riding Hood") }
```

imposes a binary constraint and is satisfied when the pair of variables (?book, ?title) can be bound to one of the pairs in the body. In the query corpus, it is used almost exclusively for unary conditions, that is, to test if the value of a single variable is in a given set of constants. We therefore also treat Values similar to Filter and Bind. That is, we call a Values block k-ary if there is a k-ary tuple following the Values keyword. Unary and binary Values subqueries are suitable for graph shape analysis and we extend the graph with an edge for each binary Values constraint. Again, higher arity Values conditions are considered in Section 7.

Service is used extensively in Wikidata queries, most commonly for Wikidata's labeling service. For this reason, Bielefeldt et al. [7] ignore the labeling service entirely in their co-occurrence analysis of SPARQL features. Similarly to before, we say that a Service subquery S is k-ary if it contains k variables. All unary or binary Service subqueries are suitable for graph analysis. When we consider the graph of patterns with Service, we add edges of the form $\{x, y\}$ for all binary Service subqueries, in which x and y are the variables. Higher arity Values conditions are considered in Section 7.

By C2RPQ+ we denote the entire fragment that uses And, Optional, Filter, property paths, Bind, Service, and Values and that is suitable for graph analysis. In total, this amounts to 176,679,495 (30,371,003) robotic and 342,576 (169,861) organic queries, which make up 85.22% (88.39%) and 51.50% (65.22%) of the robotic and organic queries, respectively.

6.5 The Set of Shape Analysis Experiments

We have analyzed the graphs of 8 fragments of queries, namely CQ, C2RPQ, CQ_F, C2RPQ_F, CQ_OF, C2RPQ_OF, $\text{CQ}_{\text{OF}+}$, and C2RPQ+. All fragments were analyzed across three dimensions: *robotic* versus *organic*, *Valid* versus *Unique*, and *with constants* versus *without constants*. In the analyses without constants, we removed all nodes in the graphs that originated from IRIs or literals.[12] Furthermore, for all fragments, we analyzed the time-out (TO) queries separately from the others (OK). This results in 64 separate runs of the shape analysis.

We classified the graphs of queries into the following set of shapes, inspired by [9]. A *chain (of length k)* is graph that is isomorphic to the undirected graph with edges $\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_k\}$. As an edge case, we allow chains of length zero, i.e., a single node. A *chain set* is a graph in which every connected component is a chain.

A *tree* is a nonempty undirected graph such that, for every pair of nodes x and y, there exists exactly one undirected path from x to y. A *forest* is a nonempty graph in which every connected component is a tree.

A *star* is a tree for which there exists at most one node with more than two neighbors. Hence, every chain is a star.

We also recall the definition of *flowers* from Bonifati et al. A *petal* is a graph consisting of a source node s, target node t, and a set of at least two node-disjoint paths from s to t. (For instance, a cycle is a petal that uses two paths.) A *flower* is a graph consisting of a node x with three types of attachments: chains (the *stamens*), trees that are not chains (the *stems*), and *petals*. Finally, a *bouquet* is a graph in which every connected component is a flower.

6.6 Shape Classification for C2RPQ+

Due to space constraints, we cannot present our complete shape analysis, but we will show and discuss the results on the valid queries in the largest fragment, the C2RPQ+ queries. Furthermore, we give some insights about how the results change for the other fragments. The results for the valid C2RPQ+ queries are in Table 6. We note that some of the queries were empty (0.8% with constants and 2.44% after removing constants); we did not include them.

In Table 6, we see several trends that we also observed in the analysis for the other fragments. First of all, in the shapes that

[12] We do not have the space to present all these analyses, but we plan to release a full version on ArXiv as soon as the paper is accepted.

Table 6: Cumulative shape analysis of nonempty graph patterns in C2RPQ+ across the valid logs

	C2RPQ+ with constants / Valid				C2RPQ+ without constants / Valid			
Shape	*#Queries (Organic)*	*Relative % (Organic)*	*#Queries (Robotic)*	*Relative % (Robotic)*	*#Queries (Organic)*	*Relative % (Organic)*	*#Queries (Robotic)*	*Relative % (Robotic)*
node (without edges)	0	0.00%	0	0.00%	85,601	25.13%	63,048,127	36.58%
chain (length ≤ 1)	107,436	31.48%	125,277,683	71.49%	200,597	58.89%	155,126,595	89.99%
chain	207,292	60.74%	158,150,895	90.25%	259,074	76.06%	163,055,757	94.59%
star	290,665	85.17%	171,767,410	98.02%	312,128	91.63%	170,205,746	98.74%
tree	329,701	96.61%	171,885,247	98.09%	321,914	94.50%	170,243,847	98.76%
flower	335,271	98.24%	172,070,177	98.19%	323,830	95.07%	170,413,306	98.86%
chain set	209,540	61.40%	158,723,217	90.57%	273,992	80.44%	165,014,582	95.73%
forest	333,725	97.79%	172,461,994	98.41%	337,730	99.15%	172,203,761	99.90%
bouquet	339,440	99.46%	172,646,921	98.52%	339,661	99.71%	172,373,220	100.00%
tw ≤ 2	341,221	99.98%	175,240,211	100.00%	340,632	100.00%	172,375,717	100.00%
tw ≤ 3	341,268	100.00%	175,240,228	100.00%	340,632	100.00%	172,375,717	100.00%
tw ≤ 4	341,274	100.00%	175,240,237	100.00%	340,632	100.00%	172,375,717	100.00%
total	341,274	100.00%	175,240,237	100.00%	340,632	100.00%	172,375,717	100.00%

include constants, *stars are quite common*. In the C2RPQ+ queries, 85.17% (87.13%) of the organic and 98.02% (99.30%) of the robotic queries are stars. The number of acyclic queries is even larger: consistently over 99% when constants are absent. As opposed to valid queries, in the timeout logs, the number of cyclic queries significantly increases. For organic CQs, for instance, the number of cyclic queries goes up to about 10%. [13] This number decreases somehow for more complex query fragments, but is still about 7.5% for the unique C2RPQ+ queries (both organic and robotic) and around 3%–4% if constants are removed. Together with the observation from Section 4 that valid queries contain 2.58 triples on average, whereas valid timeout queries have 5.65 triples on average, this suggests that cyclicity and query size play an important role in efficient query evaluation.

The logs strongly confirm a hypothesis that is often stated in theoretical research: the cyclic queries in practical applications are only *mildly cyclic*, i.e., tw $\leq k$ for small values of k. This means that database queries typically do not have large k-cliques encoded in their shape, but remain tree-like. Indeed, the largest treewidth we found in the entire logs was four, for which we found 15 (5) queries.

7 HYPERGRAPH ANALYSIS

Hypergraphs generalize graphs in the sense that they allow more than two nodes per edge. As such, the queries that were not suitable for graph shape analysis in Section 6 because they either went beyond *graph patterns* or used Filter, Service, Bind, or Values constraints with arity three or more can be considered here. This amounts to a total of 1,915,550 (1,229,035) $\mathrm{CQ}_{\mathrm{OF+}}$ queries that were not yet analysed in Section 6.

We keep the restriction on well-designed Optional constructs with interface width 1, since for these queries, there still is a correlation between the cyclicity of the hypergraph and complexity of query evaluation [4]. We do not consider queries with property paths in the hypergraph analysis.

A *hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is its finite set of nodes and $\mathcal{E} \subseteq 2^{\mathcal{V}}$ is a set of *hyperedges*. The *canonical hypergraph of a SPARQL pattern P* is defined as $\mathcal{E} = \{X \mid$ there is a triple pattern t in P such that X is the set of blank nodes and variables appearing $t\}$ and $\mathcal{V}$ is the union of the nodes in the edges in $\mathcal{E}$.

[13] We omit the exact numbers due to space constraints.

Using the tool `detkdecomp` [12], we analyzed the hypergraphs of all $\mathrm{CQ}_{\mathrm{OF+}}$ queries for which the Optional constructs are well-designed and have interface width one. Overall, we found 590,005 (273,947) remaining queries with hypertreewidth two. All others new queries had hypertreewidth one.

8 QUERY SIMILARITY SEARCH

Massive query logs are valuable sources of information as long as they are made usable for exploration and analysis. In order to improve the usability of the logs, we designed a query similarity search facility that will be released with our code and whose principles and effectiveness are discussed here. Let us first observe that string similarity search is not suitable for Wikidata logs since they have been modified by an anonymization process prior to their release. Hence, one cannot find a query by simply looking at its exact original string or a sufficiently approximate version of it, by applying string edit distance (SED), for instance. So, how can we enable query similarity search in Wikidata query logs?

Given the differentiations of these logs in terms of the structural features of the underlying queries as explained in Section 6, we opted for using the query structure as a yardstick to compare queries in terms of their similarity. This choice solves several problems when comparing queries: comments, prefixes, and variable names do not affect the structure of a query and can be excluded from the computation of query similarities. Furthermore, the anonymization of the Wikidata logs renames all variable names to names with ascending numbers (?var1, ?var2, etc., see Figure 3). Therefore, if a new variable in the query is placed before other variables, it will shift the names of all subsequent variables in the anonymized version, even though they were unchanged in the original query. This could potentially lead the similarity engine to believe that a large modification took place even though it did not. The anonymisation also removes all prefixes and inlines them, hence the impact of changing a single IRI could become large depending on the length of the IRI in case of adoption of a SED measure.

To enable a structural search on Wikidata queries, we translate the abstract syntax tree (AST) of a query into a tree structure as follows. Recall the definition of a *SPARQL query Q* in Section 3 as a triple (*query-type*, *pattern P*, *solution-modifier*). The *query-type* and the *solution-modifier* of the query respectively become

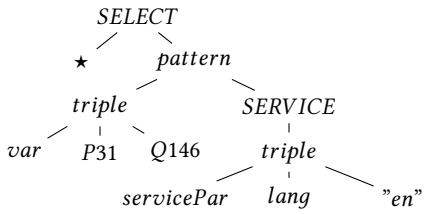

Figure 4: Abstract Syntax Tree of an example query

the root and a leaf node in the tree (with a value for the latter), whereas the *pattern* P is a subtree instantiated as a triple t with three leaves ($\langle s, p, o \rangle$), or a property path pattern, or a subtree rooted in And, Filter, Union, Optional, Graph whose respective patterns are also subtrees, or, recursively, the subtree of a query Q.

Figure 4 exemplifies the translation of the following Wikidata query from the logs into the AST.

```
SELECT ?item ?itemLabel WHERE { ?item wdt:P31 wd:Q146.
  SERVICE wikibase:label
   { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en"}}
```

Notice that in Figure 4, we ignore the actual instantiation of the variable ?item by naming it *var* and we also ignore the variables that are used in the target list of the Select clause for the same reason. We also removed the namespace prefixes. We now measure query similarity by using a tree edit distance (TED) measure. Each node in the above example is given a weight contributing to the TED. In particular, in the above AST for the example query, all nodes have equal weight (= 1 to not introduce bias). Adjustments of the weights are of course possible targeting a specific user requirement. The above example AST tree has thus a total weight of 12, this being the sum of the weight of its nodes. Within the logs, we found a query almost identical to the above example that only changed the object wd:Q146 to wd:Q11538. This renaming a single node in a triple would result in a similarity of 11/12 = 0.92.

Renaming two nodes in a triple would result in a similarity of 0.83. Adding a single triple anywhere, for example, outside the Service clause, would result in a total weight of 16. Such a change amounting to a total weight of 4 would result in a similarity of 12/16 = 0.75. Conversely, this can be seen as as a removal by starting with a tree exhibiting the additional triple and deleting it.

As for the implementation of the TED algorithm, we opted for an available implementation of the APTED algorithm [23, 24], because the implementation was easy to embed into our software, offered the option to use different weights for nodes, and yielded good performance results in our tests.

To calculate the similarity between an initial query Q and a new query Q_2, we take the size of the largest AST and divide it with the calculated TED. We invert this value by subtracting it from 1.0 in order to measure similarity instead of dissimilarity. When collecting matches in a collection of queries (within the same log file), we used a minimum threshold of 0.75. A threshold of 0.5 would mean that two queries are equally similar and dissimilar, so the step to 0.75 is right in the middle between this and being classified as identical (which corresponds to similarity equal to 1). Based on manually inspecting samples, the threshold of 0.75 already seemed to be rather lenient, and our method based on TED yielded more predictable results for us than another method we experimented

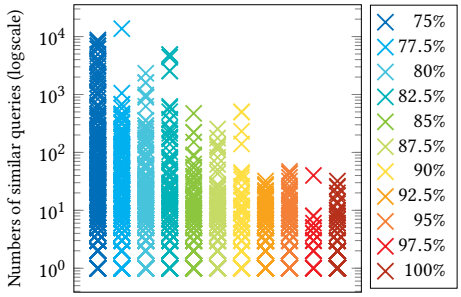

Figure 5: Number of queries that exhibit a similarity degree between 75% and 100%.

with based on SED. For example, a typical Wikidata entity has a string length of about 36. Even if all triples are changing, this can result in almost a 0.9 string similarity for the entire query, because the prefixes of entities are always the same.

In order to measure the structural similarity of Wikidata queries, we focused on the OrganicOK query logs by considering their Unique version only.[14] Since we needed seed queries for which we want to find similar queries, we considered the well known set of online Wikidata examples from [32]. We extracted[15] 356 out of the available 412 queries, which were the ones that could be parsed by Jena. It turns out that matches for most online example queries are found in the Wikidata logs analysed in our paper. Only 28 queries could not be matched, while the rest had at least one match above the threshold of 0.75.

Figure 5 shows in logscale the number of similar queries that we found in the OrganicOK Unique query logs in the order of ascending similarity. We can observe some outliers in several buckets in the scatterplot, showing that these queries have higher number of similar counterparts with respect to their pairs in the same bucket. The entire log could be scanned for the similarity search quite efficiently in less than 20 mins (without any optimization.)

9 RELATED WORK

Several empirical studies of SPARQL query logs have been conducted in the literature investigating statistical features, such as such as occurrences of triple patterns and types of queries [2, 14, 22, 28] along with recent studies on more complex features of the queries, such as structural features and complexity of the queries [9, 26]. However, the analysis of Wikidata queries has been limited in the past due to their unavailability. Bonifati et al. [9] only focused on the online 308 Wikidata queries in their large corpus and, more recently, Malyshev et al. [20] and Bielefeld et al. [7] were the first to analyze a massive collection of Wikidata queries. Malyshev et al. first introduced the Wikidata SPARQL service WDQS and analyzed basic characteristics of Wikidata queries related to their usage in this service spanning from SPARQL feature prevalence and correlation to annotations and language distributions. They also isolated the robotic and organic queries, on whose classification we rely in this paper (modulo a small correction highlighted

[14]The same analysis can be applied to the other logs but is probably less interesting for the present discussion.
[15]On August 23, 2018. Jena version same as in Section 2.

in Section 2). This classification considered as organic the queries issued by a browser and robotic the remaining queries. They further corrected the first number by identifying high-volume traffic from a single source (more than 2,000 entries), in which case the queries would be considered as machine-generated. They also identified the C2RPQs fragment, i.e. the largest fragment encountered so far of conjunctive 2-way regular path queries on which we focus in this paper for an in-depth analysis. To the best of our knowledge, the latter is the largest fragment of recursive graph queries available to our community. In this paper, we address the first large-scale structural analysis of this gigantic query collection and articulate it as follows. We provide a structural classification of real-world property paths, on the first large set of property paths relevant for Wikidata, and 57x larger than the set considered mainly for DBpedia in Bonifati et al. [9]. We also investigate the shape of C2RPQs, which could not be possible with the classification for only conjunctive queries (CQs) in Bonifati et al. [9]. The occurrences of property paths in the latter corpus is negligible with respect to the size of the C2RPQ fragment considered in our work (smaller by two orders of magnitude).

10 CONCLUSIONS AND LESSONS LEARNED

We have presented an in-depth analysis of the recently released Wikidata query logs and highlighted the presence of their most prominent query fragment, i.e. C2RPQs. This fragment corresponds to highly complex recursive queries with joins and property paths. Apart from simple counting measures on this fragment, we have focused on tailoring property path analysis and shape analysis to these queries, whereas previous work merely looked at conjunctive queries (CQs) in DBpedia logs [9]. Even though we agree with Bielefeldt et al. [7] on the difficulty of obtaining stable observations from these query logs due to massive presence of robotic traffic, we discovered several similarities, like low hypertreewidth and structure of property paths seem to be relatively consistent between the present study and previous work.

But what about the distinguishing features of these logs compared to previous logs and the findings that were possible on this newly discovered fragment, i.e. C2RPQs? Entirely new observations were made across the distinction in terms of Valid and Unique logs, further segmented into RoboticOK and OrganicOK, the addition of timeout logs never analysed before, which led us to add interesting dimensions to the analysis.

We now revisit the research questions posed at the end of Section 1 and we succinctly report the main findings of our analysis for those questions. About the distributions of sizes, we found out that the RoboticOK queries are less skewed in terms of sizes than the OrganicOK queries and this also applies to OrganicTO and RoboticTO queries, that are also inherently more complex in terms of sizes than the RoboticOK plus OrganicOK queries.

Next, property paths occur in these logs 57x more than in previously analyzed logs, which made us focus on the most representative query fragment C2RPQs. We noticed that a bigger variety of property path classes again occurs with Organic queries compared to Robotic queries. Still, we see that almost all property paths match paths of *any possible length* once a certain minimum length is exceeded. (Typical such expressions are ab^* or ab^*c.) Expressions that do not satisfy this property, like $(ab)^*$, appear as well, but are less prominent. In fact, the most occurring types of property paths found were similar to those found in an earlier study [9], which may allow us to identify subclasses of expressions that are important in practice. Such a discovery might spur interesting query processing and query optimization questions around C2RPQs, which were not addressed for the much simpler fragments of CQs. For instance, landmarking indexes have been introduced for one of the prominent classes of Robotic queries (A^*) in Valstar et al. [30], but also the other prominent classes need attention when designing indexes for C2RPQs.

In our analysis, we also addressed the question on the prominence of CQs and C2RPQs in these logs compared to other logs, thus bringing to the surface the most occurring recursive fragment of C2RPQs enriched with And, Optional, Filter, Bind, Service and Values (C2RPQ+). We ran a shape classification with and without constants for several fragments ranging between the class of CQs and C2RPQ+ by considering/excluding constants. Here, we see that *star shapes* and *tree-like queries* are very common. The shape analysis with or without constants also led us to identify shifts in the shape classes due to removal of constants that are worth looking at. For instance, if we remove constants from the graph shape of queries in the logs, many (25% organic and 36% robotic) disintegrate to a single node. Constants have been disregarded in the study of C2RPQs, where indexing techniques have mainly considered the labeled paths as key index terms. The combination of indexing techniques looking at constants and labeled paths could thus be a direction to pursue in future studies on indexing structures and index maintenance for these queries [8]. The timeout queries are also interesting because they are on average larger in size and more cyclic than the valid queries.

Concerning treewidth and hypertreewidth, the logs strongly confirm a hypothesis that is often stated in theoretical research: the cyclic queries in practical applications are only *mildly cyclic*. This means that database queries typically do not have large k-cliques encoded in their shape, but remain tree-like. This observation is in line with the data from a previous study [9].

Next, a novel Wikidata-specific query similarity search allows to efficiently navigate the query logs starting from an initial query that the user has at her disposal. This search improves the usability of the Wikidata query logs for both recursive and non-recursive queries and is valuable for identifying subsets of similar queries on which further assessment is possible.

After pre-processing, i.e., computing the Valid and Unique data sets, our entire analysis (except query similarity search, whose performance has been separately measured) takes 12 hours on a 24 core machine with a 2.6 GHz CPU and 128 GB RAM.

We believe that isolating complex query fragments and studying suitable sophisticated metrics is valuable for the community and may lead to further studies and assessment of these logs. We believe that this work can serve as a basis for researchers to find further interesting fragments of queries to study; we report what we see in the logs. However, one should always keep in mind that we are looking at *specific query logs*. It cannot be concluded from this study that a given fragment, operator, or type of query is *not* interesting to study.

REFERENCES

[1] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. 2012. Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *World Wide Web Conference (WWW)*. 629–638.

[2] Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. 2011. An Empirical Study of Real-World SPARQL Queries. *CoRR* abs/1103.5043 (2011).

[3] Guillaume Bagan, Angela Bonifati, and Benoît Groz. 2013. A trichotomy for regular simple path queries on graphs. In *Principles of Database Systems (PODS)*. 261–272.

[4] Pablo Barceló, Markus Kröll, Reinhard Pichler, and Sebastian Skritek. 2018. Efficient Evaluation and Static Analysis for Well-Designed Pattern Trees with Projection. *ACM Trans. Database Syst.* 43, 2 (2018), 8:1–8:44.

[5] Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. 2015. Efficient Evaluation and Approximation of Well-designed Pattern Trees. In *Principles of Database Systems (PODS)*. 131–144. https://doi.org/10.1145/2745754.2745767

[6] Adrian Bielefeldt, Julius Gonsior, Larry Gonzalez, Markus Krötzsch, and Stanislav Malyshev. 2018. Wikidata SPARQL Logs. https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en.

[7] Adrian Bielefeldt, Julius Gonsior, and Markus Krötzsch. 2018. Practical Linked Data Access via SPARQL: The Case of Wikidata. In *Linked Data on the Web (LDOW)*. http://events.linkeddata.org/ldow2018/papers/LDOW2018_paper_4.pdf.

[8] Angela Bonifati, George H. L. Fletcher, Hannes Voigts, and Nikolay Yakovets. 2018. *Querying Graphs.* Morgan & Claypool Publishers.

[9] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An Analytical Study of Large SPARQL Query Logs. *PVLDB* 11, 2 (2017), 149–161.

[10] Chandra Chekuri and Anand Rajaraman. 1997. Conjunctive Query Containment Revisited. In *International Conference on Database Theory (ICDT)*. 56–70.

[11] DARQL. 2019. https://github.com/PoDMR/darql/.

[12] Detkdecomp. 2018. https://github.com/daajoe/detkdecomp. Visited on September 10th, 2018.

[13] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. The complexity of acyclic conjunctive queries. *J. ACM* 48, 3 (2001), 431–498.

[14] Xingwang Han, Zhiyong Feng, Xiaowang Zhang, Xin Wang, Guozheng Rao, and Shuo Jiang. 2016. On the statistical analysis of practical SPARQL queries. In *International Workshop on Web and Databases (WebDB)*. 2.

[15] Stever Harris and Andy Seaborne. 2013. *SPARQL 1.1 query language.* Technical Report. World Wide Web Consortium (W3C). https://www.w3.org/TR/2013/REC-sparql11-query-20130321.

[16] Oren Kalinsky, Yoav Etsion, and Benny Kimelfeld. 2017. Flexible Caching in Trie Joins. In *International Conference on Extending Database Technology (EDBT)*. 282–293.

[17] Phokion G. Kolaitis and Moshe Y. Vardi. 1998. Conjunctive-Query Containment and Constraint Satisfaction. In *Symposium on Principles of Database Systems (PODS)*. 205–213.

[18] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. 2013. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.* 38, 4 (2013), 25:1–25:45.

[19] Katja Losemann and Wim Martens. 2013. The complexity of regular expressions and property paths in SPARQL. *ACM Trans. Database Syst.* 38, 4 (2013), 24:1–24:39.

[20] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. 2018. Getting the Most out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *International Semantic Web Conference (ISWC)*. 376–394.

[21] Wim Martens and Tina Trautner. 2018. Evaluation and Enumeration Problems for Regular Path Queries. In *International Conference on Database Theory (ICDT)*. 19:1–19:21.

[22] K. Möller, M. Hausenblas, R. Cyganiak, S. Handschuh, and G. Grimnes. 2010. Learning from linked open data usage: Patterns & metrics. In *Web Science Conference (WSC)*.

[23] Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient Computation of the Tree Edit Distance. *ACM Trans. Database Syst.* 40, 1, Article 3 (March 2015), 40 pages.

[24] Mateusz Pawlik and Nikolaus Augsten. 2015. Tree edit distance: Robust and memory-efficient. *Information Systems* 56 (08 2015). https://doi.org/10.1016/j.is.2015.08.004

[25] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009), 16:1–16:45. https://doi.org/10.1145/1567274.1567278

[26] François Picalausa and Stijn Vansummeren. 2011. What are real SPARQL queries like?. In *International Workshop on Semantic Web Information Management (SWIM)*. 1–7.

[27] Alessandro Piscopo. 2018. Wikidata: A New Paradigm of Human-Bot Collaboration? *arXiv preprint arXiv:1810.00931* (2018).

[28] Muhammad Saleem, Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. LSQ: The Linked SPARQL Queries Dataset. In *International Semantic Web Conference (ISWC)*. 261–269.

[29] Sunburst diagram. 2019. Sunburst diagram. https://podmr.github.io/darql/property-sunburst/.

[30] Lucien D. J. Valstar, George H. L. Fletcher, and Yuichi Yoshida. 2017. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. In *International Conference on Management of Data (SIGMOD)*. 345–358.

[31] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[32] Wikidata. 2018. https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples.

[33] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *International Conference on Very Large Data Bases (VLDB)*. 82–94.