Characterizing Robotic and Organic Query in SPARQL Search Sessions

Xinyue Zhang[1], Meng Wang[1,2] (✉), Bingchen Zhao[3], Ruyang Liu[1], Jingyuan Zhang[1], and Han Yang[4]

1. Southeast University, Nanjing, China
2. Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China
3. Tongji University, Shanghai, China
4. Peking University, Beijing, China

{zhangxy216,meng.wang}@seu.edu.cn

Abstract. SPARQL, as one of the most powerful query languages over knowledge graphs, has gained significant popularity in recent years. A large amount of SPARQL query logs have become available and provided new research opportunities to discover user interests, understand query intentions, and model search behaviors. However, a significant portion of the queries to SPARQL endpoints on the Web are robotic queries that are generated by automated scripts. Detecting and separating these robotic queries from those organic ones issued by human users is crucial to deep usage analysis of knowledge graphs. In light of this, in this paper, we propose a novel method to identify SPARQL queries based on session-level query features. Specifically, we define and partition SPARQL queries into different sessions. Then, we design an algorithm to detect loop patterns, which is an important characteristic of robotic queries, in a given query session. Finally, we employ a pipeline method that leverages loop pattern features and query request frequency to distinguish the robotic and organic SPARQL queries. Differing from other machine learning based methods, the proposed method can identify the query types accurately without labelled data. We conduct extensive experiments on six real-world SPARQL query log datasets. The results demonstrate that our approach can distinguish robotic and organic queries effectively and only need 7.63×10^{-4} seconds on average to process a query.

Keywords: SPARQL · Session Search · Query Classification

1 Introduction

With the rapid development of Semantic Web technologies, more and more data are published as knowledge graphs in Resource Description Framework (RDF) [11] triple form (*subject*, *predicate*, *object*). SPARQL [9], as one of the most widely used query languages for accessing knowledge graphs, has become the de-facto standard in this context. Currently, there are approximately 1.5×10^{11} RDF triples from different domains[1] that can be explored by 557 SPARQL

[1] `http://linkeddata.org/`

endpoints on the Web[2]. As a result, numerous SPARQL query logs are generated every day and have recently become available for researchers to discover user interests, understand query intentions and model search behaviors [15, 17].

Motivations: Conducting extensive analysis of massive SPARQL logs is challenging. One of the main problems is that there is a significant portion of queries to SPARQL endpoints that are robotic queries. Robotic queries are usually generated and issued by automated scripts or programs for index size inferring, data crawling, or malicious attacking, while organic queries imply the real information need of human users. Raghuveer [15] pointed out that 90% of queries in USEWOD dataset [3] are requested by less than 2% no-human users. Similarly, in DBpedia[3] SPARQL query log dataset, 90% queries are provided by only 0.4% automated software programs. It indicates that robotic queries dominate organic ones in terms of volume and query load. Therefore, it is crucial to pre-process query logs by detecting and separating robotic queries from organic queries before diving into deep analysis works.

Most of existing methods [4, 15] on distinguishing between robotic and organic query are mainly based on agent names recorded in SPARQL logs [4, 15] and query request frequency [15]. However, each of them has disadvantages. For agent names, it is simple and effective to select organic queries from trusted agents, but the trusted agent list needs to be manually specified and is not always available. Following the specification of Apache's log format[4], agent names will be recorded on 400 error and 501 error only. Besides, smart crawlers can fake agent names by adding them to the request header. For query request frequency, how to determine an appropriate threshold is annoying. Therefore, recognizing the different types of queries only by the agent name or frequency is not enough. Moreover, several machine learning based methods [10, 19] have been proposed to detect robotic queries in conventional search engines. However, they rely on user demography features and sufficient labelled training data, which are usually missing in SPARQL search scenarios.

Solutions: Given the above observations, in this paper, we propose a framework to classify robotic and organic queries by detecting features of robotic queries in SPARQL session-level. Specifically, we organize sequences of queries as sessions which are defined considering the time and semantic constraints. Then, according to three types of loop patterns that distribute in robotics queries, *i.e.*, single intra loop pattern, the sequence of intra loop pattern, and inter loop pattern, we design algorithms to detect each pattern. Our loop detection algorithm is a training-free process which focuses on detecting characteristic of robotic queries and has high efficiency with a complexity of $O(nlogn)$ where n presents the session length. Finally, we implement a pipeline method that takes query request frequency and loop pattern features into consideration to distinguish robotic and organic queries. To guarantee the high precision for organic queries, a rule has been specially set to relax the constraint of robotic queries, *i.e.*, if

[2] `https://sparqles.ai.wu.ac.at/availability`
[3] `https://wiki.dbpedia.org/`
[4] `http://httpd.apache.org/docs/current/mod/mod_log_config.html`

one session that comes from one user is detected with loop patterns, then all the sessions of the same user will be classified into robotic queries. Moreover, our method can provide the explanation for each identified query (*i.e.*, filtered by frequency, or the specific loop pattern).

Contributions: The contributions of this paper are summarized as follows:

– We propose an efficient and simple pipeline method in SPARQL session-level to distinguish between organic and robotic queries.
– We design a new training-free algorithm that can accurately detect loop patterns that is an important characteristic for robotic queries, with a complexity of $O(nlogn)$ where n is the session length.
– We conduct extensive experiments on six real-world SPARQL query log datasets. The results indicate that our approach is effective and efficient.

Organization: The remainder of this paper is organized as follows. Sec 2 presents basic SPARQL query log analysis. The details of our method (including preliminary, loop pattern detection algorithm, and query classifying method) are described in Sec 3. In Sec 4, we show experiments on real-world SPARQL queries to demonstrate the effectiveness and efficiency of our method. Sec 5 discusses related work. Finally, conclusions are presented in Sec 6.

2 SPARQL Query Log Analysis

Before we design our query detection algorithm, we first collect real-world SPARQL query logs and present basic analysis.

2.1 Datasets

We use data collected from six different SPARQL endpoints: affymetrix[5], dbsnp[6], gendr[7], goa[8], linkedspl[9], and linkedgeodata[10]. The first five datasets are a part of Bio2Rdf [2] which is a bioinformatic RDF cloud. The linkedgeodata [20] makes the information collected by the OpenStreetMap project [7] available as an RDF knowledge graph. All these SPARQL logs have been modified into RDF format like LSQ [18], which makes them easy to be analyze.

In our collected data, every SPARQL query and execution is identified by an unique id. One query can have multiple executions. We recognize different users by their encrypted IP address. The basic information about the datasets in this paper can be found in Tab 1. *Queries* column indicates the number of

[5] `http://affymetrix.bio2rdf.org/sparql`
[6] `http://dbsnp.bio2rdf.org/sparql`
[7] `http://gendr.bio2rdf.org/sparql`
[8] `http://goa.bio2rdf.org/sparql`
[9] `http://linkedspl.bio2rdf.org/sparql`
[10] `http://linkedgeodata.org/sparql`

Table 1: Statistics of SPARQL query logs.

Data Source	queries	executions	users	begin time	end time
affymetrix	618,796/630,499	1,782,776/1,818,020	1,159	2013-05-05	2015-09-18
dbsnp	545,184/555,971	1,522,035/1,554,162	274	2014-05-23	2015-09-18
gendr	564,158/565,133	1,369,325/1,377,113	520	2014-01-16	2015-09-18
goa	630,934/638,570	2,345,460/2,377,718	1,190	2013-05-05	2015-09-18
linkedgeodata	651,251/667,856	1,586,660/1,607,821	26,211	2015-11-22	2016-11-20
linkedspl	436,292/436,394	756,806/757,010	107	2014-07-24	2015-09-18

queries without parse error and the number of all the queries. *Executions* column presents executions without parse error and the number of all the executions. We also list the number of users and the time interval of the data.

2.2 Preliminary Analysis

We perform preliminary analysis about query distributions over users and time span, as well as query template repetitions.

Distribution of Queries Executed by Users: As mentioned above, many prior works [18, 15] have noticed that most SPARQL queries are provided by few no-human users, and we also find the similar phenomenon in our data. We first group queries by users and then sort users by the number of queries they execute. Then we calculate how many users contribute to 95% executions at least. Results can be found in Tab 2. In terms of the number of executions, 95% executions are contributed by very few users (less than 7%) in all the datasets, and less than 0.5% in the sum of all datasets.

Table 2: 95% executions are contributed by α% users.

dataset	affymetrix	dbsnp	gendr	goa	linkedspl	linkedgeodata	all
α	1.47	3.65	1.54	1.60	1.87	6.80	0.40

Table 3: The percentage(β%) of unique templates

dataset	affymetrix	dbsnp	gendr	goa	linkedspl	linkedgeodata	all
β	0.25	0.28	0.16	0.20	0.67	0.19	0.28

Distributions of Queries, Users, and Time Span together: We associate users with the number of queries they submit and the time span of these submitted queries for each user, as illustrated in Fig 1. In terms of the number of users, most users execute 1 $\sim$ 80 queries within 1 hour.

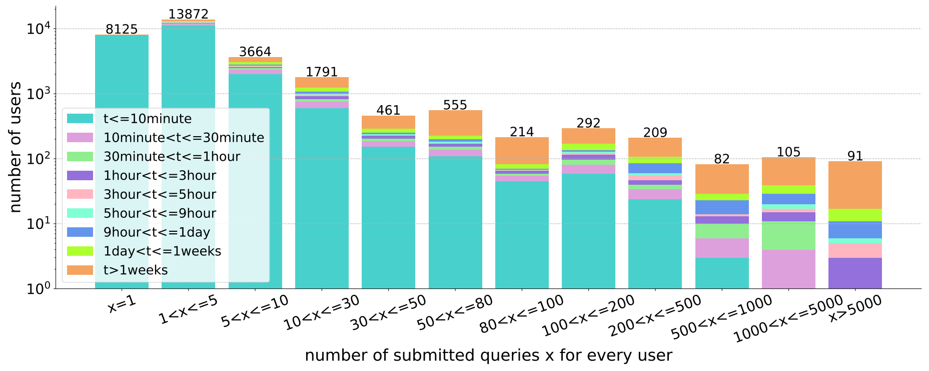


Fig. 1: Distributions of the number of submitted queries x and time span t of the submitted queries for every user. The X-axis indicates different intervals about the number of submitted queries, Y-axis means how many users are in this interval. Different colors in the bar mean different time span of these users.

Query Template Repetition: As mentioned by Raghuveer [15], robotic queries tend to use fixed query templates. We extract the query template for every query in our data. We match the extracted query templates to calculate the percentage of unique templates over all queries, which means that the lower the number is, the more repetitions of templates are in queries. The query templates are extracted by replacing IRI, variable and literal with *_IRI_, _VAR_, _LIT_* respectively like [15]. We calculate the similarity between two templates based on string edit distance by fuzzywuzzy[11]. The query template repetition results of six datasets are reported in Tab 3. We find that the percentage of unique templates behind queries is less than 0.7%. For all the data sets except linkedgeodata, the number of unique templates is about 0.3% of the number of all queries. The results indicate that the large repetitive query templates exist in real-world queries.

3 Our Method

In this section, we present the definition of SPARQL query session based on time and semantic constraints, as well as descriptions of three loop patterns, which are important features of robotic queries and characterized by the distribution of query templates in a given SPARQL query session. Then, we design a loop pattern detection algorithm to capture the loop features of robotic queries. Finally, we implement a pipeline method that leverages query request frequency and loop pattern features to solve the organic and robotic query identifying problem.

[11] `https://pypi.org/project/fuzzywuzzy/`

3.1 Preliminaries

Definition 1. ***Term(q): term set of one query.*** *All variables and specific terms (*i.e. *RDF IRIs) used in the query are included in the term set.*

Definition 2. ***Session.*** *Considering a sequence of queries* $q_1, q_2, q_3 \cdots q_n$[12], *which is ordered by time and executed by one user. We define a SPARQL query sequence as a* ***session*** *if it satisfies the following two constraints:*

- *If we use* $time(q)$ *to represent the time when query* q *is executed, this sequence of queries satisfies* $time(q_n) - time(q_1) < time\_threshold$.
- *For any continuous query pair* (q_i, q_{i+1}) *in this sequence, it satisfies* $term(q_i) \cap term(q_{i+1}) \neq \emptyset$.

In this paper, we set *time_threshold* to 1 hour. The reason why we include variables in the term set is that users usually do not change variable names they use in a query. If two continuous queries executed by one user share one variable, we can infer that two queries have some potential correlations and should be included in the same session. Next, we introduce three types of loop patterns in the SPARQL session-level.

Single Intra Loop Pattern: Robotic queries often come from a loop in automated scripts or programs trying to collect enough information to satisfy their uses. In these sessions, the structure of queries remains the same, but variables, literals or IRIs are changing. (1) In some cases, machines want to collect all the information about one specified *subject*, then in queries, only *predicates* are changing as shown in below Example1. (2) For cases in which machines want to find out the same information shared by some *subjects*, the *subjects* are changing, as shown in Example2. (3) If a machine wants to collect the *subjects* with certain types or values, then only *objects* are changing (see Example3). (4) There are also cases in which the numeric values in SPARQL constraint operators *FILTER*, *OFFSET* and *LIMIT* and the string values in *REGEX* functions are changing.

```
Example1: predicate change
{ ?s  <http://bio2rdf.org/affymetrix_vocabulary:x-flybase>  ?o}
{ ?s  <http://bio2rdf.org/affymetrix_vocabulary:x-omim>  ?o }

Example2: subject change
{<http://linkedgeodata.org/triplify/node2957398896> rdfs:label ?label}
{<http://linkedgeodata.org/triplify/node1885439658> rdfs:label ?label}

Example3: object changing
{?item rdf:type <http://www.openlinksw.com/schemas/rdfs/TechArticle#this>}
{?item rdf:type <http://wordnet.okfn.gr/resource/synset-noun-2> }
```

[12] We only consider queries without parse errors and merge the same queries in adjacent positions. For instance, a sequence $[0, 1, 1, 1, 2]$ (in which $0, 1, 2$ means the query id) can be processed to $[0, 1, 2]$

All the cases described above remain the structure of the original SPARQL query, and change one or more variables, IRIs and literal values. This is to say, the query templates behind queries in this kind of loop are the same. This is the so-called *single intra loop pattern*. If we use 0 to represent the template index, $'+'$ means appearing one or more times, then the single intra loop pattern can be expressed by [0+].

Sequence of Intra Loop: In our dataset, excepting the single intra loop introduced above, we also notice there are *sequences of intra loops*. This type of loop pattern shows up when, for example, the machine gets one target attribute for all the *subjects*, then queries for another attribute. If we use numbers to represent template index, $'+'$ means appearing one or more times, then this loop pattern can be expressed by $[0+1+\cdots]$.

Inter Loop Pattern: Another type of loop pattern is *inter loop pattern*, which is used, for instance, to query all the features for one *subject*, then change to another *subject*. Using the same method as above, inter loop pattern can be expressed by $[(01\cdots)+]$. Notice 0,1 here can be a single query or a intra loop.

3.2 Loop Pattern Detection Algorithm

In this section, we introduce our loop pattern detection algorithm, as shown in Fig 2. In a nutshell, the algorithm we implement contains the following steps:

- **Step1:** ***Generate templates***, we organize the queries as sessions (Such as *QuerySeq* in Fig 2) defined in Sec 3.1, and replace each query in the original session with its corresponding template index to generate a sequence of template index.
- **Step2:** ***Merge the same items in adjacent positions***, we merge the continuous same items and generate a sequence of template index without repetitions in adjacent positions (*i.e.* $TmpltSeqWoRep$). An example of this step is shown in Fig 2.
- **Step3:** ***Is it a single intra loop?***, we detect a single intra loop pattern which can be expressed as [0+]. Therefore, if $TmpltSeqWoRep$ only contains one template index, then this session has intra loop pattern.
- **Step4:** ***Is it a sequence of intra loop?***, we detect the *sequences of intra loop* which have the order like $[0+1+\cdots]$. We recognize such pattern by calculating the percentage of $len(TmpltSeqWoRep)$ and $len(QuerySeq)$ and regard sessions with this value lower than $thre1$ as sessions with *sequence of intra loop patterns*. We design this step because that, if merging the same items in adjacent positions can let the length of the session shrink to lower than a threshold, there must be so many repetitions in adjacent positions. The set of $thre1$ is provided in Sec 3.2.
- **Step5:** ***Is it a inter loop***, we detect *inter loops* which has the pattern like [(01)+]. Details about this function can be found in *Inter Loop Detection* section below.

If a session contains one of the patterns described above, then we think queries of this session can be classified into robotic queries. All the thresholds

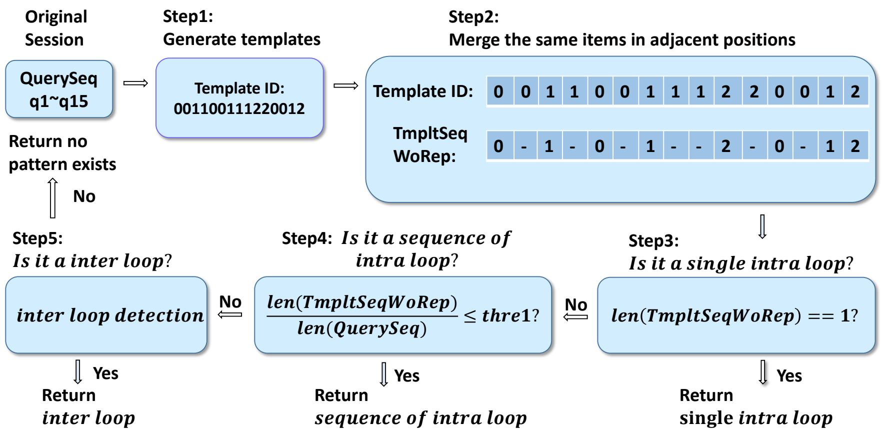

Original Session
QuerySeq q1~q15
Step1: Generate templates
Template ID: 001100111220012
Step2: Merge the same items in adjacent positions
Template ID: 0 0 1 1 0 0 1 1 1 2 2 0 0 1 2
TmpltSeq WoRep: 0 - 1 - 0 - 1 - - 2 - 0 - 1 2
Step3: Is it a single intra loop?
len(TmpltSeqWoRep) == 1?
Yes
Return single intra loop
No
Step4: Is it a sequence of intra loop?
len(TmpltSeqWoRep) / len(QuerySeq) ≤ thre1?
Yes
Return sequence of intra loop
No
Step5: Is it a inter loop?
inter loop detection
Yes
Return inter loop
No
Return no pattern exists

Fig. 2: Overview of loop pattern detection algorithm.

mentioned in Algorithms are determined by experiments in *Thresholds Setting* section below.

Inter Loop Detection: As described in Sec 3.1, the inter loop pattern can be expressed by [(01...)+]. We detect this pattern by calculating the maximum subsequence which loops over the entire session. We use a *Queue* to store this subsequence. A detailed example is provided by Fig 3. Scanning the input ($TmpltSeqWoRep$) from left to right, add the item that do not exist in *Queue* into *Queue* (step1 $\sim$ 2 in Fig 3). For items that are already in *Queue*, the subsequence beginning from specific item in $TmpltSeqWoRep$ should be matched against subsequence in *Queue*. The subsequence can match over the sequence in *Queue* from the beginning (step3 $\sim$ 4 and step6 $\sim$ 8) or from the middle. Also, *Queue* can be extended like step5. Notice that Fig 3 is only the first step of *Detect inter loop* function. The percentage of $len(Queue)$ and $len(QuerySeq)$, presents in what extent intra loop pattern exists. Therefore, if this value is lower than $thre2$, then we think a inter loop pattern exists in this session.

Thresholds Setting: In order to find 2 thresholds mentioned in Sec 3.2, we extract $3,000$ sessions in all the data randomly to find different features in sessions with different lengths. We use len_ori, $len1$, $len2$ to indicate $len(QuerySeq)$, $len(TmpltSeqWoRep)$, $len(Queue)$ in the following sections. len_ori, is just the length of original session; $len1$ means the length of session after removing the continuous same template index; $len2$ is the length of maximum subsequence which appears in a session, corresponding to the length of *Queue* in Fig 3. We consider three kinds of features:

– Distribution of $len1/len\_ori$ in sessions with different lengths, which can present the distribution of intra loop, both *single intra loop* and *sequence intra loop pattern.*

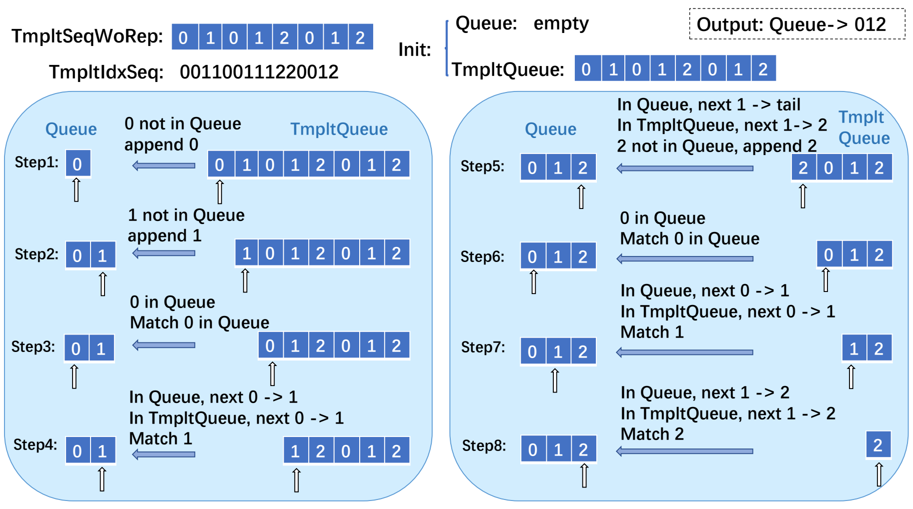
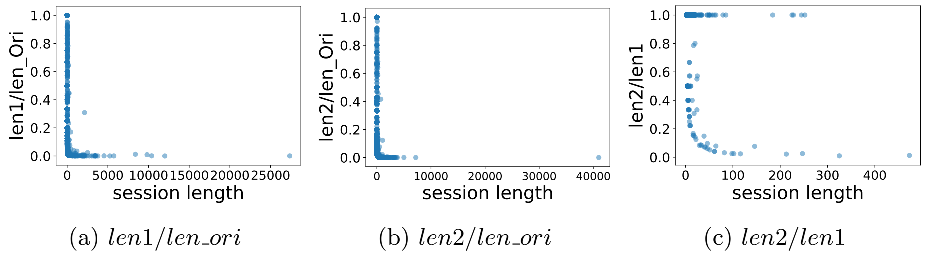


Fig. 3: An example of *inter loop detection*



(a) $len1/len\_ori$ (b) $len2/len\_ori$ (c) $len2/len1$

Fig. 4: Distribution of $len1/len\_ori$, $len2/len\_ori$ and $len2/len1$.

- Distribution of $len2/len\_ori$ in sessions with different lengths, which can present the distribution of *intra loop* and *inter loop pattern*.
- Distribution of $len2/len1$ in sessions with different lengths, which can present the distribution of *inter loop*, because $len2$ is computed based on $TmpltIdxWoRep$.

The Distribution of three features can be seen in Fig 4. In terms of $len1/len\_ori$, sessions with lengths more than 100 are almost 0, which indicates there are lots of *intra loops*. On the contrary, in shorter sessions with lengths less than 100, distribution of $len1/len\_ori$ is different. The turning point of $len1/len\_ori$ in Fig 4a is about 0.1, therefore, we set $thre1$ to 0.1. Using the same method, according to Fig 4b, we set $thre2$ to 0.1.

Comparing three figures in Fig 4, we can conclude that most of the sessions with lengths longer than 100 contain *intra loop patterns*. In sessions with lengths of $100 \sim 500$, there are a few *inter loop patterns* existing.

Complexity: Considering the process in Fig 2, assuming that the length of original session is n, step3 and step4 have the complexity of constant. The complexity of step2 is linear, and the complexity for step1 and step5 which contains the operation of finding an item in an ordered list is $O(nlogn)$. Therefore, the complexity of our loop pattern detection algorithm is $O(nlogn)$.

3.3 Robotic and Organic Query Classification Pipeline Method

We design a pipeline method to classify robotic and organic queries by leveraging query request frequency and loop patterns, which contains the following steps:

1) ***Frequency Test:*** For a query sequence ordered by time and generated by one user, we check the query request frequency first. For every query in this sequence, we create a time window in 30 minutes and if the number of queries in this window is more than 30, we infer the query request frequency of this sequence is too high and all the queries generated by this user are detected as robotic queries.

2) ***Session Generation:*** We organize query sequences as sessions following the definition we introduce in Sec 3.1.

3) ***Loop Pattern Detection Algorithm:*** Using algorithm described in Sec 3.2, we detect loop patterns based on SPARQL query sessions.

Considering the number of organic queries is very small, we think the recall of robotic query classification is more important. Therefore, we set a rule: if one of the sessions of one user can be detected as a loop pattern, all the queries of the same user will be classified into robotic queries. Note that, the agent name constraint can also be added into this pipeline before the *Frequency Test*. Usually, some browser-related agent names are selected as a sign for organic queries.

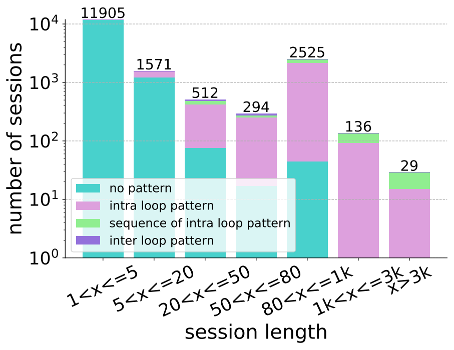
4 Experiments

To scrutinize the effectiveness and efficiency of the proposed method, We conduct experiments on six real-world datasets. We first evaluate the loop pattern detection algorithm and its average runtime. Then, we validate the effectiveness of our pipeline method to classify robotic and organic queries, as well as the efficiency for a query and a session.

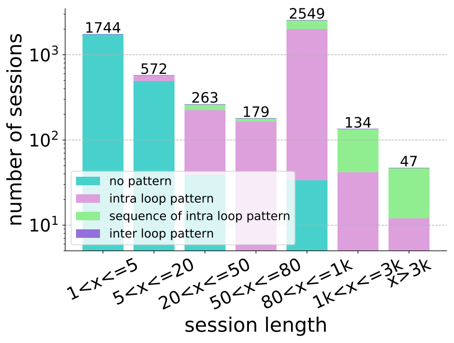
4.1 Loop Pattern Detection

We detect three loop patterns mentioned in Sec 3.1 using our loop pattern detection algorithm. Results shown in Fig 5 and Fig 6 indicate our algorithm can recognize all the sessions with lengths more than 1,000 in 5/6 datasets and most sessions with lengths of $80 \sim 1,000$. For dbsnp, gendr and linkedspl, there are loop patterns distributed in sessions with lengths of $50 \sim 80$.

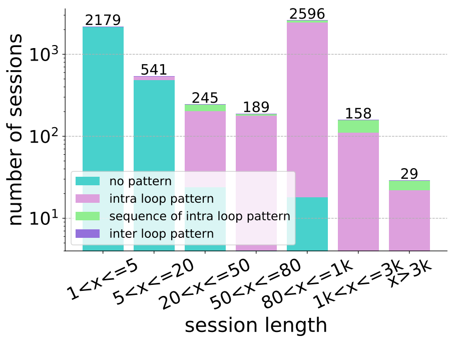
Also, in different datasets, the distribution of different patterns is a little different. The most common loop pattern in all the datasets is the *single intra loop pattern*. The second common loop pattern is the *sequence of intra loop pattern*, which appears in dbsnp in particular. Besides, in linkedgeodata, there is a considerable number of *inter loop patterns*. Most sessions with lengths more



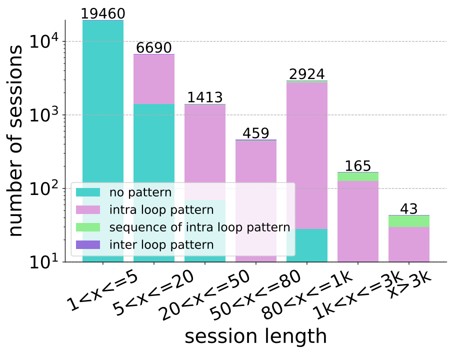
(a) affymetrix



(b) dbsnp



(c) gendr



(d) goa

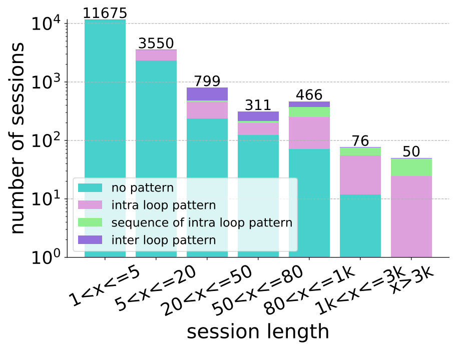
Fig. 5: Loop pattern distribution in affymertrix, dbsnp, gendr and goa.

than 80 can be detected, which illustrates our algorithm can capture features of robotic queries. For the efficiency, experiments on linkedspl dataset show our algorithm can process a query in 2.37×10^{-4} seconds, and a session in 0.001 seconds on average.

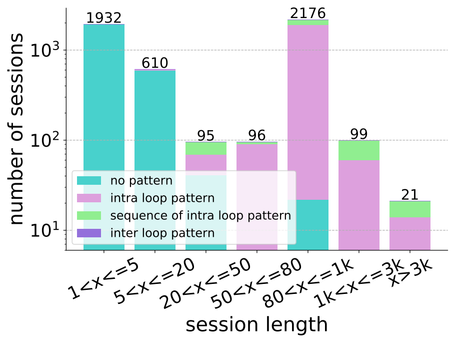
4.2 Robotic and Organic Query Classification

Our pipeline method utilizes the query request frequency and loop patterns to identify robotic and organic queries. The classification results are reported in Tab 4. We also list the number of robotic queries filtered by different constraints. Query request frequency can filter out the most robotic queries and our Loop Pattern Detection Algorithm can filter out a considerable number of robotic queries. Even though the number of queries filtered by loop pattern detection is smaller than the number of queries filtered by frequency, it is still a fairly big number considering the number of organic queries. Taking gendr as an example, the overall number of organic queries is 1262, but the number of queries filter by loop pattern is 1214, which will disturb analysis work if they are mixed up.

Furthermore, considering there is no ground truth in robotic and organic query classification tasks, we visualize the distribution of queries requested to



(a) linkedgeodata



(b) linkedspl

Fig. 6: Loop pattern distribution in linkedgeodata and linkedspl.

Table 4: Classification Results.

dataset	Robotic Queries Count	Organic Queries Count	Robotic Queries Filtered By Loop	Robotic Queries Filtered By Freq
affymetrix	990,684	7,659	1,108	989,576
dbsnp	1,191,001	2,146	55	1,190,946
gendr	981,549	1,262	1,214	980,335
goa	1,430,506	1,827	624	1,429,882
linkedspl	745,810	1,230	376	745,434
linkedgeodata	1,480,573	18,380	10,160	1,470,413

endpoints at different times within one day and use the difference of distributions in robotic and organic queries to evaluate the effectiveness of our methods. Results can be seen in Fig 7 and Fig 8. We only show two canonical distributions here. The rest datasets are similar to these two distributions. The distributions in both linkedgeodata and gendr for organic queries follow a strong daily rhythm. Like [4], with most activities happening during the European and American day and evening. This indicates a direct human involvement. For robotic queries, most of them are uniformly distributed. An interesting thing here is that, in $1:00 \sim 3:00$ in gendr, we check queries in this time interval and find most of the queries are very likely to come from an automated script that examines the availability of endpoint every day. These kinds of queries are hard to remove for two reasons: 1) They do not have a high request frequency. 2) They do have diversity in the session-level. But we can notice their existence by visualization we present here, and we can remove such queries manually. Besides, the experiment shows that our pipeline method can process a query in 7.63×10^{-4} seconds and a query sequence from one user in 5.33 seconds on average.

5 Related Work

SPARQL Log Analysis: SPARQL log analysis can provide rich information in many aspects. Many prior works [21, 1, 14, 8, 13] have focused on the analysis

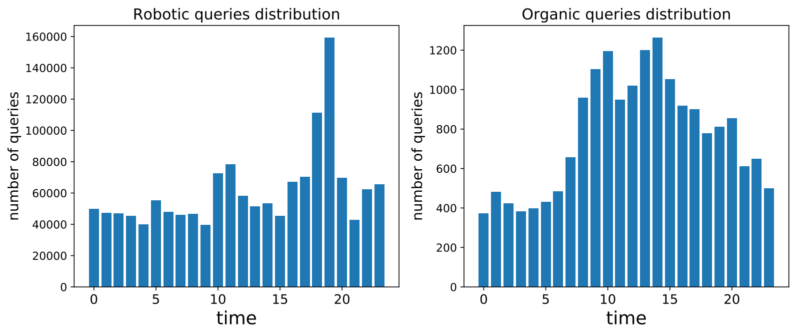


Fig. 7: Query Request Time (UTC) Distribution of the linkedgeodata.

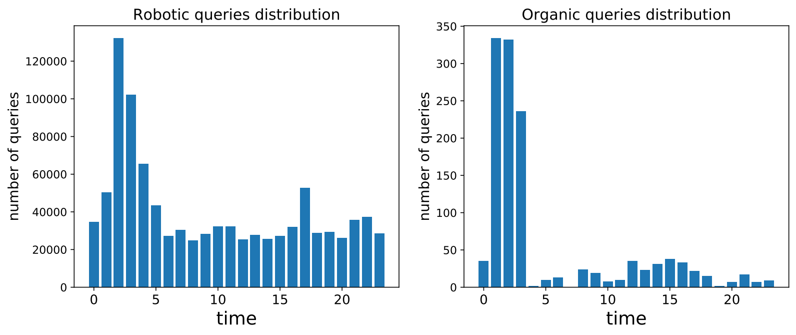


Fig. 8: Query Request Time (UTC) Distribution of gendr.

of SPARQL query logs. They analyze SPARQL queries from statistical features (*i.e.* occurrences of triple patterns) and shape related features (*i.e.* occurrences of conjunctive and non-conjunctive patterns). However, these works mainly analyse SPARQL query in isolation. Features between queries have not been fully analysed. Currently, similarity between queries in a query sequence which is from the same user and ordered by time (in [15], this sequence is called as *session*) has been noticed by [14, 5, 15]. The similarity feature has been utilized in query augmentation based on the analysis of previous (historic) queries [12, 16, 23]. In [16], authors define *session* based on the definition of [15] but add a 1-hour time window constraint. Our work moves onward by adding a semantic constraint on the definition of *session*. Raghuveer [15] introduce *intra* and *inter loop patterns* which are characteristics of robotic queries from session viewpoint. Then, they evaluate the prevalence of these patterns in USEWOD dataset [3] by loop detection technique they design. However, the method they introduce is quite simple and can not satisfy the need to distinguish between robotic and organic queries. In light of this, we classify loop patterns more carefully and give a specific definition of these patterns. Furthermore, according to the features of each pattern, we design an algorithm to detect loop patterns, which can be used in the robotic and organic query classification scenario.

Robotic and Organic Query Classification: The need to distinguish between machines and humans in SPARQL search is recognized by [15, 17, 4]. Rietveld *et al.* [17] find organic queries and robotic queries have very different features. In [15], robotic queries are recognized by query request frequency and the agent names. Bielefeldt *et al.* [4] are the first to introduce an idealised view of organic and robotic queries. They separate wikidata [22] SPARQL query logs into organic queries and robotic queries mainly by manually specified agent lists. They have published this classified dataset. Based on this dataset, Bonifati *et al.* [6] analyse different features of both queries. However, as we mentioned above, distinguishing robotic and organic queries by agent names and query request frequency has drawbacks. In this paper, we consider an important characteristic, *i.e.*, loop pattern, and design a pipeline method for robotic and organic queries classification problem leveraging query request frequency and loop pattern. Experiments on six real-world SPARQL query logs indicate that our method is more effective and efficient.

6 Conclusion

In this paper, we propose a novel method to distinguish robotic and organic queries based on SPARQL session-level query features. We first organize queries as sessions. Then, we design an algorithm to detect loop patterns, which is an important characteristic of robotic queries. Furthermore, we implement a pipeline method to separate robotic queries from organic queries by leveraging query request frequency and loop patterns. Our method does not require user demography features and sufficient labelled training data. The effectiveness and efficiency of our method has been validated by experiments on six real-world SPARQL query log datasets.

Acknowledgement

This work was supported by National Science Foundation of China with Grant Nos. 61906037 and U1736204; National Key Research and Development Program of China with Grant Nos. 2018YFC0830201 and 2017YFB1002801; the Fundamental Research Funds for the Central Universities.

References

1. Arias, M., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world sparql queries. arXiv preprint arXiv:1103.5043 (2011)
2. Belleau, F., Nolin, M.A., Tourigny, N., Rigault, P., Morissette, J.: Bio2rdf: towards a mashup to build bioinformatics knowledge systems. Journal of Biomedical Informatics **41**(5), 706–716 (2008)
3. Berendt, B., Hollink, L., Hollink, V., Luczak-Rösch, M., Möller, K., Vallet, D.: Usewod2011: 1st international workshop on usage analysis and the web of data. In: The 20th International Conference on World wide web. pp. 305–306 (2011)

4. Bielefeldt, A., Gonsior, J., Krötzsch, M.: Practical linked data access via sparql: The case of wikidata. In: The 11th Workshop on Linked Data on the Web. pp. 1–10 (2018)
5. Bonifati, A., Martens, W., Timm, T.: An analytical study of large sparql query logs. The VLDB Journal pp. 1–25 (2017)
6. Bonifati, A., Martens, W., Timm, T.: Navigating the maze of wikidata query logs. In: The World Wide Web Conference. pp. 127–138 (2019)
7. Haklay, M., Weber, P.: Openstreetmap: User-generated street maps. IEEE Pervasive Computing **7**(4), 12–18 (2008)
8. Han, X., Feng, Z., Zhang, X., Wang, X., Rao, G., Jiang, S.: On the statistical analysis of practical sparql queries. In: The 19th International Workshop on Web and Databases. pp. 1–6 (2016)
9. Harris, S., Seaborne, A., Prud'hommeaux, E.: Sparql 1.1 query language. W3C recommendation **21**(10), 778 (2013)
10. Kang, H., Wang, K., Soukal, D., Behr, F., Zheng, Z.: Large-scale bot detection for search engines. In: The 19th International Conference on World Wide Web. pp. 501–510 (2010)
11. Klyne, G., Carroll, J.J., McBride, B.: Resource description framework (rdf): Concepts and abstract syntax. w3c recommendation, feb. 2004 (2004)
12. Lorey, J., Naumann, F.: Detecting sparql query templates for data prefetching. In: Extended Semantic Web Conference. pp. 124–139. Springer (2013)
13. Möller, K., Hausenblas, M., Cyganiak, R., Grimnes, G.A., Handschuh, S.: Learning from linked open data usage: Patterns & metrics. In: The WebSci10: Extending the Frontiers of Society On-Line. pp. 1–8 (2010)
14. Picalausa, F., Vansummeren, S.: What are real sparql queries like? In: The International Workshop on Semantic Web Information Management. pp. 1–6 (2011)
15. Raghuveer, A.: Characterizing machine agent behavior through sparql query mining. In: The International Workshop on Usage Analysis and the Web of Data. pp. 1–8 (2012)
16. Rico, M., Touma, R., Queralt Calafat, A., Pérez, M.S.: Machine learning-based query augmentation for sparql endpoints. In: The 14th International Conference on Web Information Systems and Technologies. pp. 57–67 (2018)
17. Rietveld, L., Hoekstra, R., et al.: Man vs. machine: Differences in sparql queries. In: The 4th USEWOD Workshop on Usage Analysis and the Web of of Data. pp. 1–7 (2014)
18. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.C.N.: Lsq: the linked sparql queries dataset. In: International Semantic Web Conference. pp. 261–269. Springer (2015)
19. Shakiba, T., Zarifzadeh, S., Derhami, V.: Spam query detection using stream clustering. World Wide Web **21**(2), 557–572 (2018)
20. Stadler, C., Lehmann, J., Höffner, K., Auer, S.: Linkedgeodata: A core for a web of spatial open data. Semantic Web **3**(4), 333–354 (2012)
21. Stegemann, T., Ziegler, J.: Pattern-based analysis of sparql queries from the lsq dataset. In: International Semantic Web Conference (Posters, Demos & Industry Tracks). pp. 1–4 (2017)
22. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)
23. Zhang, W.E., Sheng, Q.Z., Qin, Y., Yao, L., Shemshadi, A., Taylor, K.: Secf: improving sparql querying performance with proactive fetching and caching. In: The 31st Annual ACM Symposium on Applied Computing. pp. 362–367 (2016)