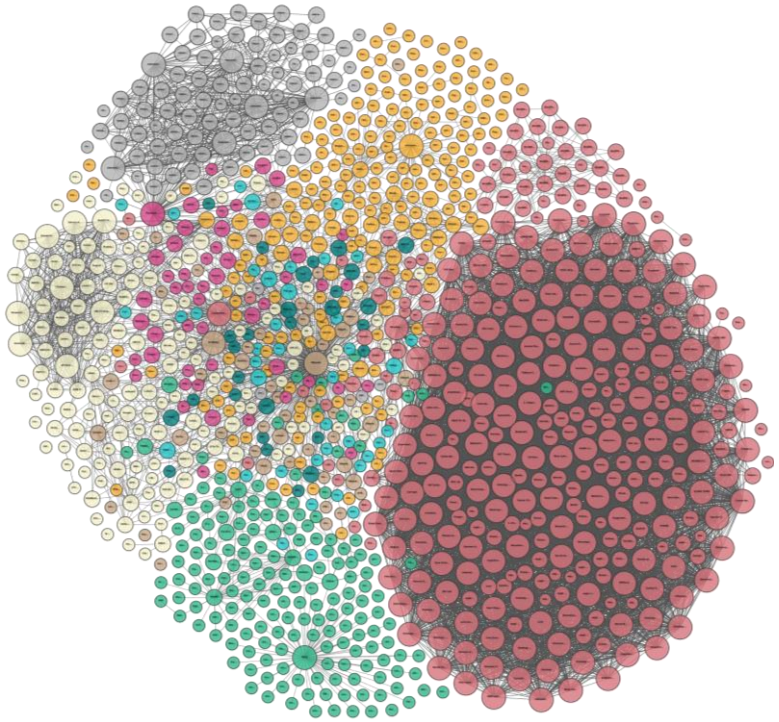
Characterizing Robotic and Organic Query in SPARQL Search Sessions

Xinyue Zhang, Meng Wang, Bingchen Zhao, Ruyang Liu,
Jingyuan Zhang, and Han Yang.

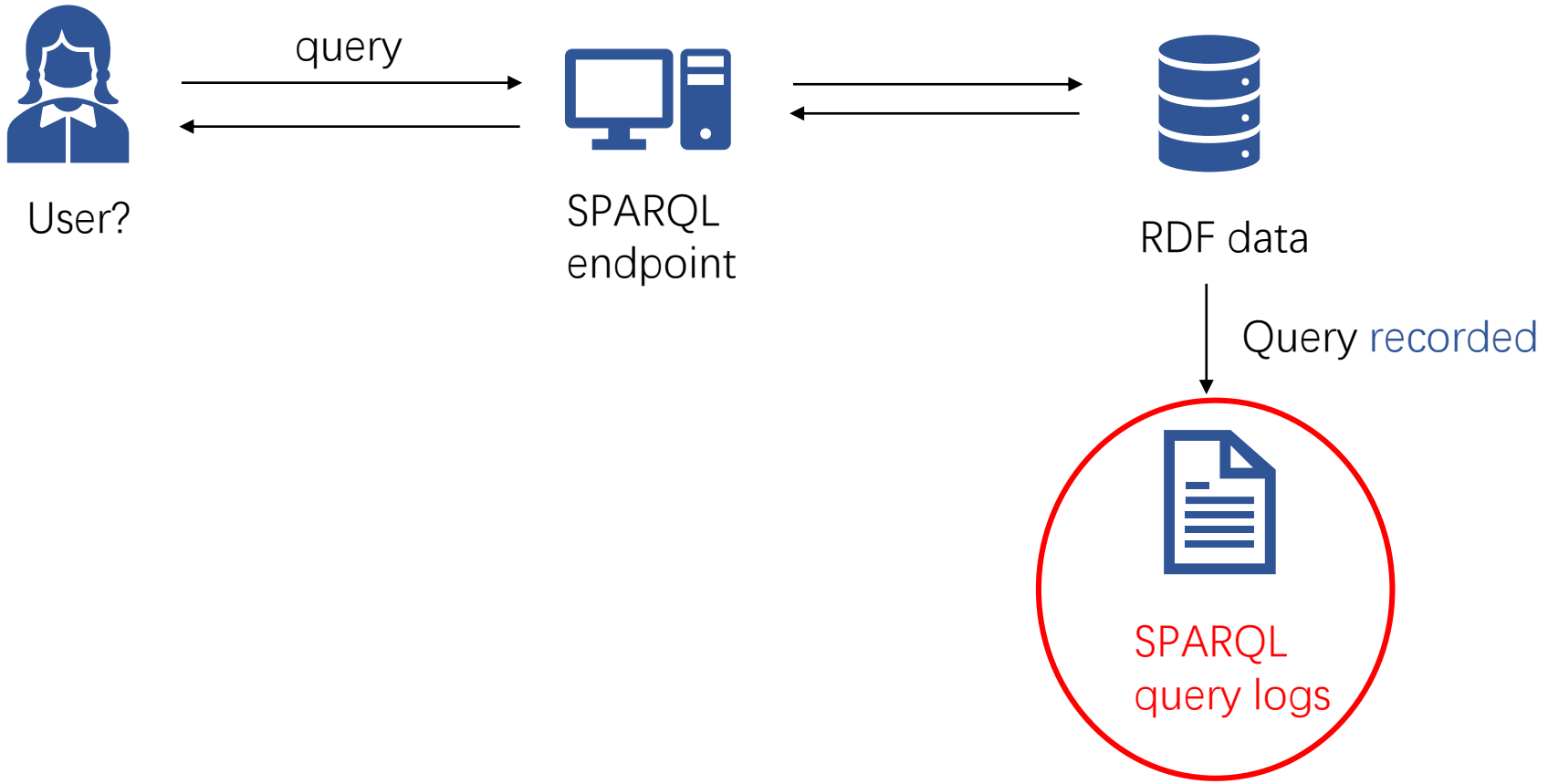
The problem we want to solve

- More RDF data can be accessed by SPARQL, a widely used query language.



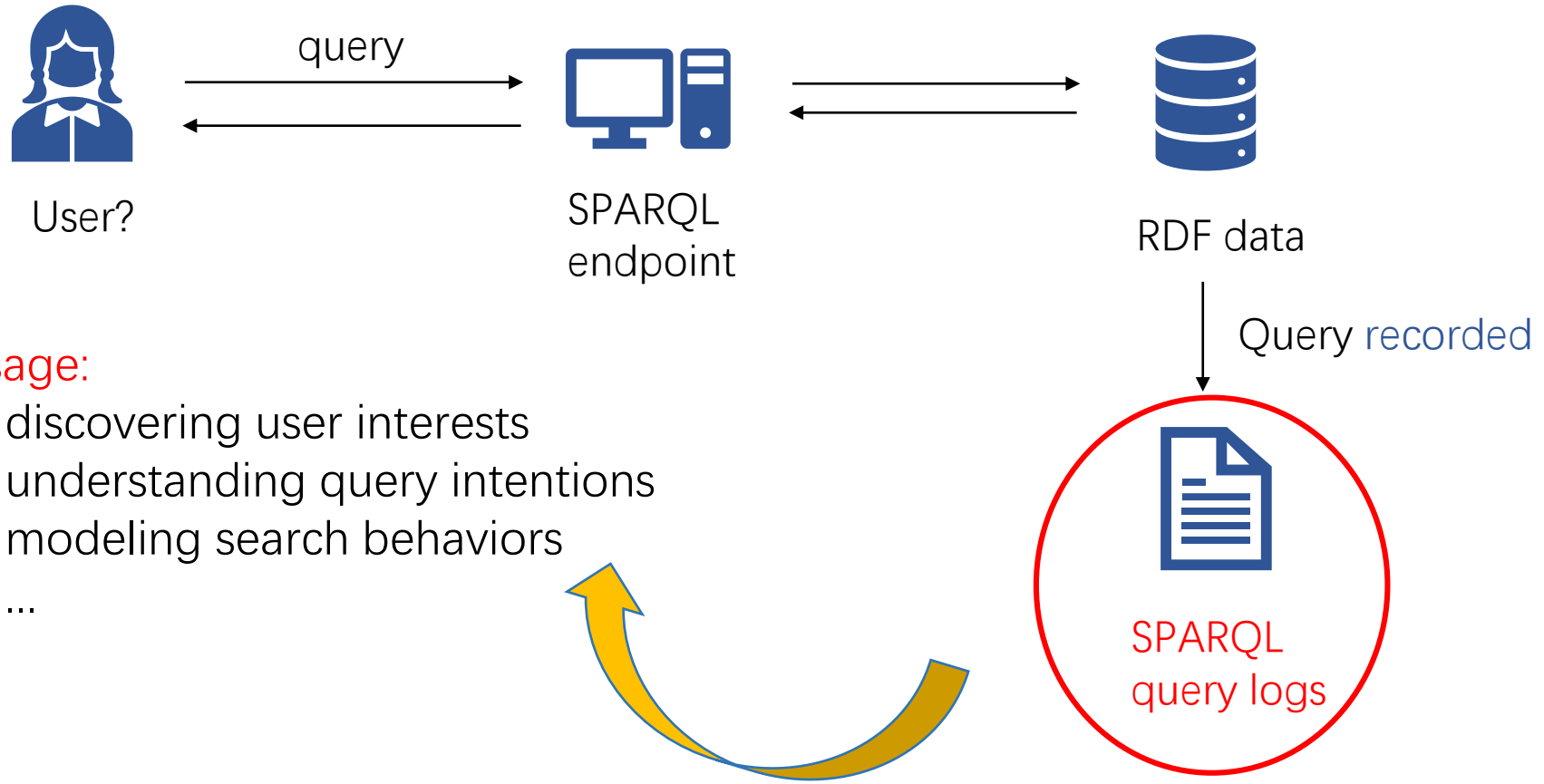
The problem we want to solve

- Numerous SPARQL queries are made every day!



The problem we want to solve

- Numerous SPARQL queries are made every day!

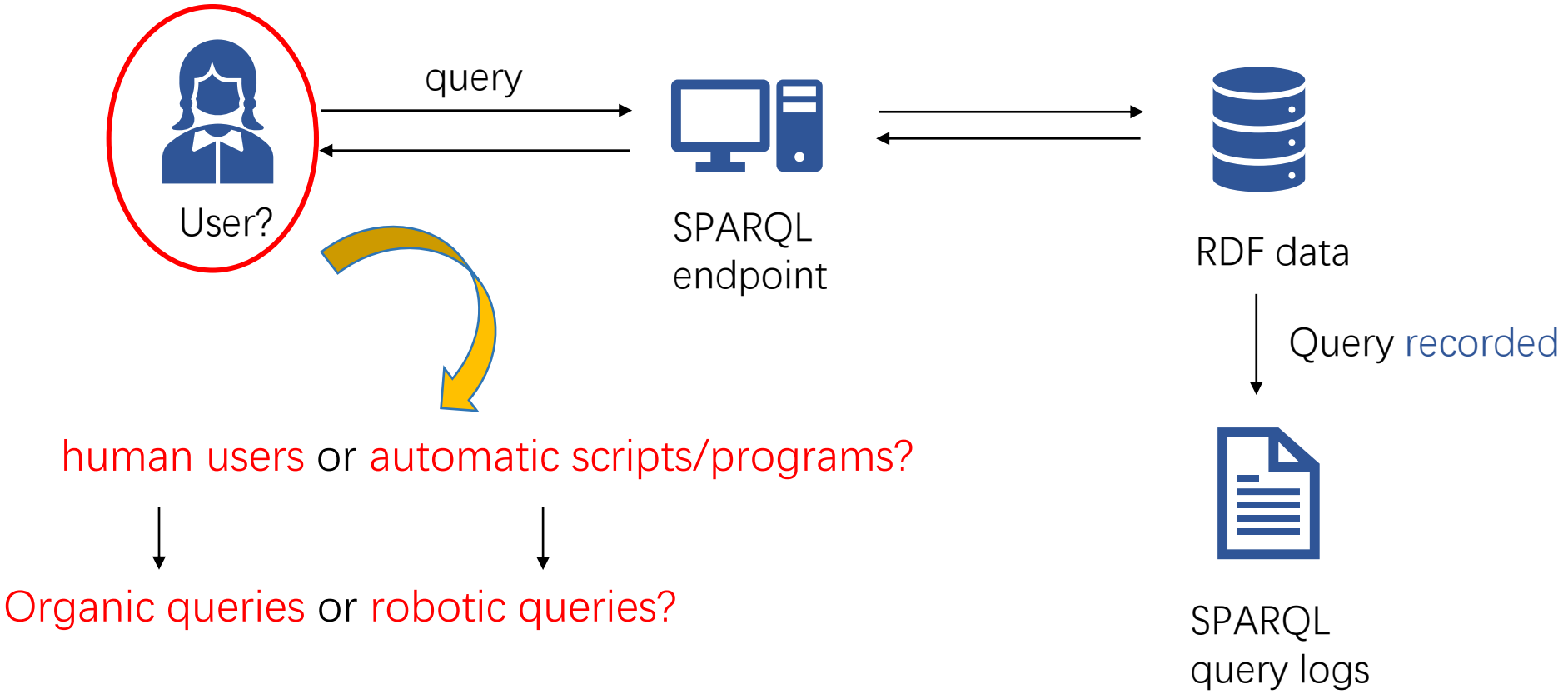


Usage:

- discovering user interests
- understanding query intentions
- modeling search behaviors
- ...

The problem we want to solve

- We only want queries submitted by human users!



Previous methods

An example log:

```
127.0.0.1 09/Jun/2014:05:13:43 -0400 SELECT * {?s ?p ?o} cu.drugbank.bio2rdf.org
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36 500 781
```

- We use the **IP address** to identify different users.

Note: More detailed information about the log format can be found in here.

Previous methods: frequency

An example log:

127.0.0.1 09/Jun/2014:05:13:43 -0400 SELECT * {?s ?p ?o} cu.drugbank.bio2rdf.org
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36 500 781

- **Timestamp** can be used to calculate the query request **frequency** of certain user.
- classify queries with a high request frequency as robotic queries.

However,

- determining an appropriate threshold is annoying.

Note: More detailed information about the log format can be found in here.

Previous methods: Agent names

An example log:

127.0.0.1 09/Jun/2014:05:13:43 -0400 SELECT * {?s ?p ?o} cu.drugbank.bio2rdf.org
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36 500 781

- **Agent names** can be used to specify human users/automatic scripts.

- Example agent names indicating human users:
  Chrome
  Mozilla
- Example agent names indicating automatic scripts:
  Java
  python

Note: More detailed information about the log format can be found in here.

Previous methods: Agent names

An example log:

127.0.0.1 09/Jun/2014:05:13:43 -0400 SELECT * {?s ?p ?o} cu.drugbank.bio2rdf.org
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36 500 781

- **Agent names** can be used to specify human users/automatic scripts.
- Wordcloud for SPARQL User Agents from [1].

Figure 3: Wordcloud for SPARQL User Agents (Dbpedia). The user agent *Java* appears 4.9 million times

Note: More detailed information about the log format can be found in here.
[1]: Characterizing Machine Agent Behavior through SPARQL Query Mining.

Previous methods: Agent names

An example log:

127.0.0.1 09/Jun/2014:05:13:43 -0400 SELECT * {?s ?p ?o} cu.drugbank.bio2rdf.org
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36 500 781

- **Agent names** can be used to specify human users/automatic scripts.

However,

- Trusted agent list needs to be manually specified.
- Not always available:
  only recorded on 400 error and 501 error only[2].
  Cannot be published because of privacy policies.
- smart crawlers can fake agent names by adding them to the request header.

Note: More detailed information about the log format can be found in here.
[2]:http://httpd.apache.org/docs/current/mod/mod_log_config.html

Our solutions

- Characterize a new feature of robotic queries: loop patterns.
- Design a loop pattern detection algorithm.
- Implement a pipeline method to distinguish robotic and organic queries.

Preliminaries

- Our SPARQL query log datasets.

Table 1: Statistics of SPARQL query logs.

dataset	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

Preliminaries

- Preliminary analysis: distribution of queries executed by users

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

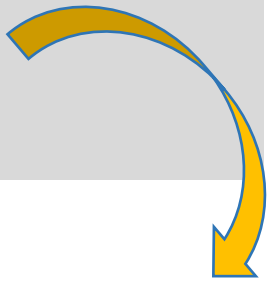
dataset	affymetrix	dbsnp	gendr	goa	linkedspl	linkedgeodata	all
α	1.47	3.65	1.54	1.60	1.87	6.80	0.40

Most queries are submitted by few users!

Preliminaries

- Preliminary analysis: query template repetition

```
SELECT *
WHERE
{
    { ?book dc10:title ?title }
      UNION
    { ?book dc11:title ?title }
}
```



Generate **template**:

- Substitute IRI with '_IRI_'.
- Substitute variable with '_VAR_'.
- Substitute literal with '_LIT_'
- Normalize the format.

```
SELECT * WHERE { { _VAR_ _URI_ _VAR_ } UNION { _VAR_ _URI_ _VAR_ } }
```

Preliminaries

- Preliminary analysis: query template repetition

Table 3: The percentage($\beta\%$) of unique templates

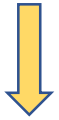
dataset	affymetrix	dbsnp	gendr	goa	linkedspl	linkedgeodata	all
β	0.25	0.28	0.16	0.20	0.67	0.19	0.28

Large repetitive query templates exist in real-world queries.

Characterizing robotic queries: loop patterns

- Robotic queries are usually generated by loops in scripts/programs.
- We use loop patterns to characterize robotic queries.

```
For fi in [QueryForAge, QueryForSchool, …]:
    For namei in [Alice, Bob, Cindy, …]: (n in total)
        fi(namei)
```



```
#1: SELECT * {Alice :age ?age}
#2: SELECT * {Bob   :age ?age}
#3: SELECT * {Cindy :age ?age}
…
#n+1: SELECT * {?school :hasStudent  Alice}
#n+2: SELECT * {?school :hasStudent    Bob}
#n+3: SELECT * {?school :hasStudent  Cindy}
…
```

Characterizing robotic queries: loop patterns

- Single intra loop pattern

```
For namei in [Alice, Bob, Cindy, …]: (n in total)
    QueryForAge(namei)
```

issue queries automatically

```
#1: SELECT * {Alice :age ?age}
#2: SELECT * {Bob   :age ?age}
#3: SELECT * {Cindy :age ?age}
…
```

Generate templates

```
#0: SELECT * {_IRI_ _IRI_ _VAR_}
#0: SELECT * {_IRI_ _IRI_ _VAR_}
#0: SELECT * {_IRI_ _IRI_ _VAR_}
…
```

[000…](n in total)

Expressed as [0+].

- 0: the template index
- +: appearing one or more times

Characterizing robotic queries: loop patterns

- Sequence of intra loop pattern

```
For fi in [QueryForAge, QueryForSchool, …]:
    For namei in [Alice, Bob, Cindy, …]: (n in total)
        fi(namei)
```

issue queries automatically

```
#1: SELECT * {Alice :age ?age}
#2: SELECT * {Bob   :age ?age}
…
#n+1: SELECT * {?school :hasStudent  Alice}
#n+2: SELECT * {?school :hasStudent    Bob}
…
```

Generate templates

```
#0: SELECT * {_IRI_ _IRI_ _VAR_}
#0: SELECT * {_IRI_ _IRI_ _VAR_}
…                                    [000…111…]
#1: SELECT * {_VAR_ _IRI_ _IRI_}
#1: SELECT * {_VAR_ _IRI_ _IRI_}
```

Expressed as [0+1+⋯].

- 0/1: the template index
- +: appearing one or more times

Characterizing robotic queries: loop patterns

- Inter loop pattern

```
For namei in [Alice, Bob, Cindy, …]: (n in total)
    For fi in [QueryForAge, QueryForSchool, …]: (m in total)
        fi(namei)
```

issue queries automatically

```
#1: SELECT * {Alice :age ?age}
#2: SELECT * {?school :hasStudent  Alice}
…
#m+1: SELECT * {Bob   :age ?age}
#m+2: SELECT * {?school :hasStudent    Bob}
…
```

Generate templates

```
#0: SELECT * {_IRI_ _IRI_ _VAR_}
#1: SELECT * {_VAR_ _IRI_ _IRI_}
…
#0: SELECT * {_IRI_ _IRI_ _VAR_}
#1: SELECT * {_VAR_ _IRI_ _IRI_}
```

[01…01…]

Expressed as [(01…)+].

- 0/1: the template index
- +: appearing one or more times

Loop Pattern Detection Algorithm

- Overview

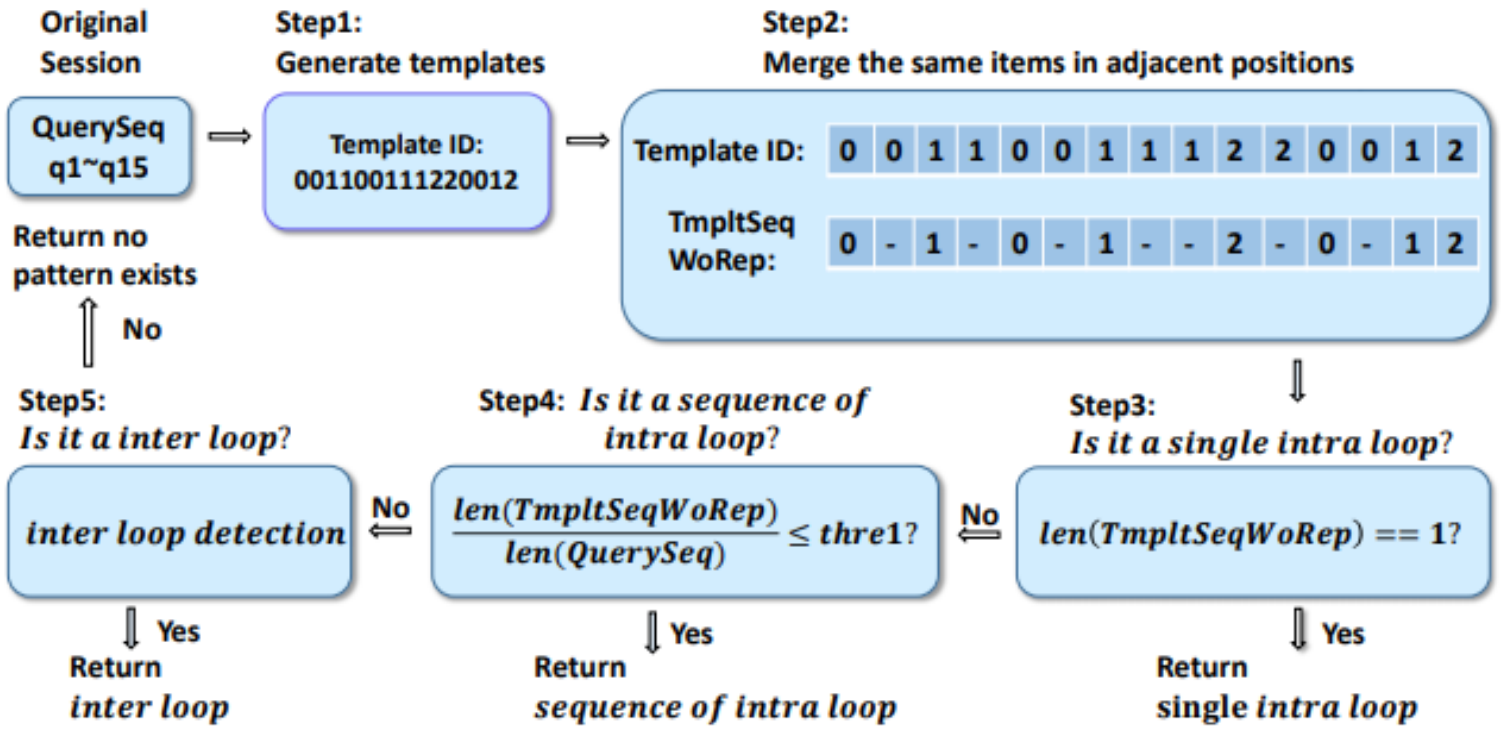
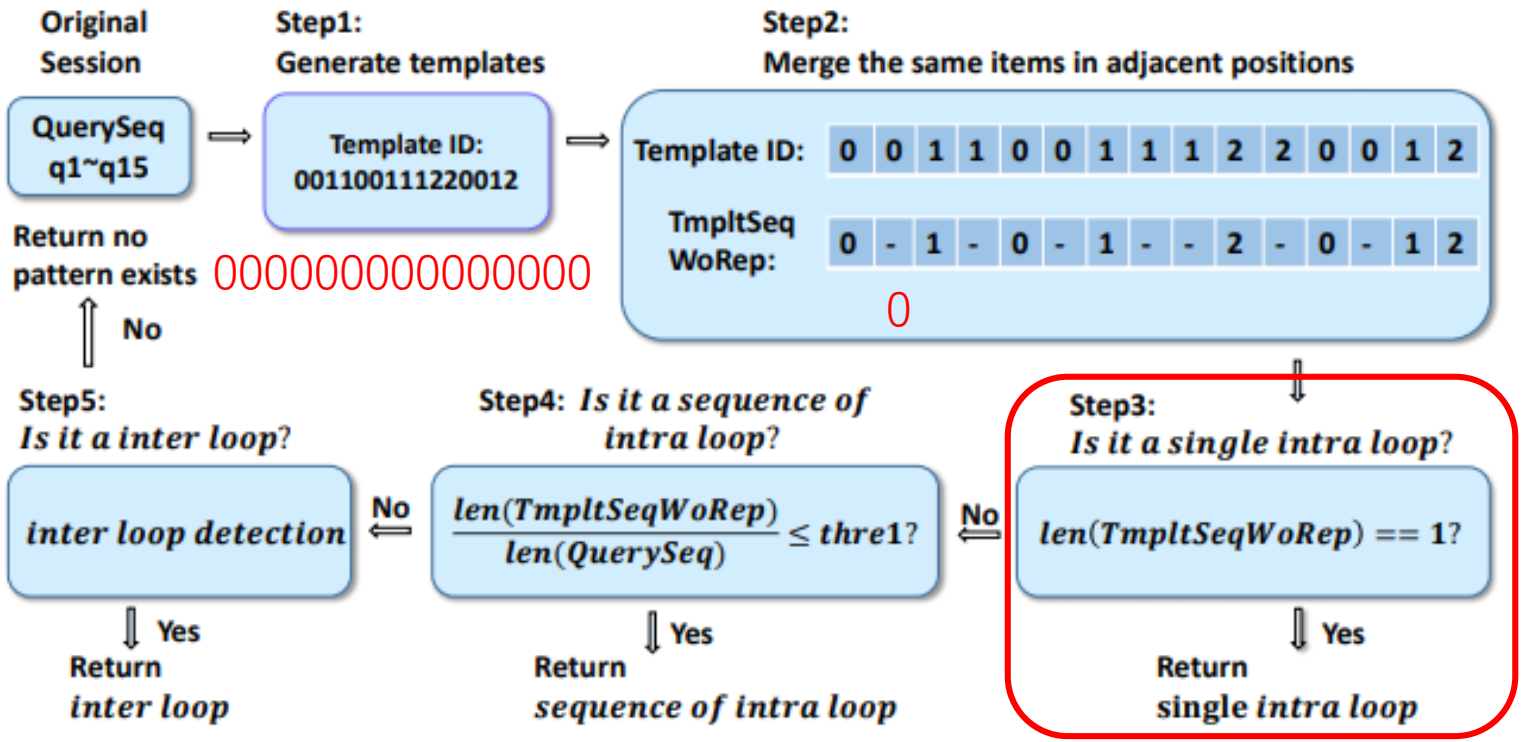


Fig. 2: Overview of loop pattern detection algorithm.

Loop Pattern Detection Algorithm

- Single intra loop pattern ([0+])



$len([0]) == 1?$

√

Fig. 2: Overview of loop pattern detection algorithm.

Loop Pattern Detection Algorithm

$$\frac{len(0123)}{len([001112223333333])} \leq thre1?$$

Must have enough repetitions in adjacent positions!

- Sequence of intra loop patterns ([0+1+⋯])

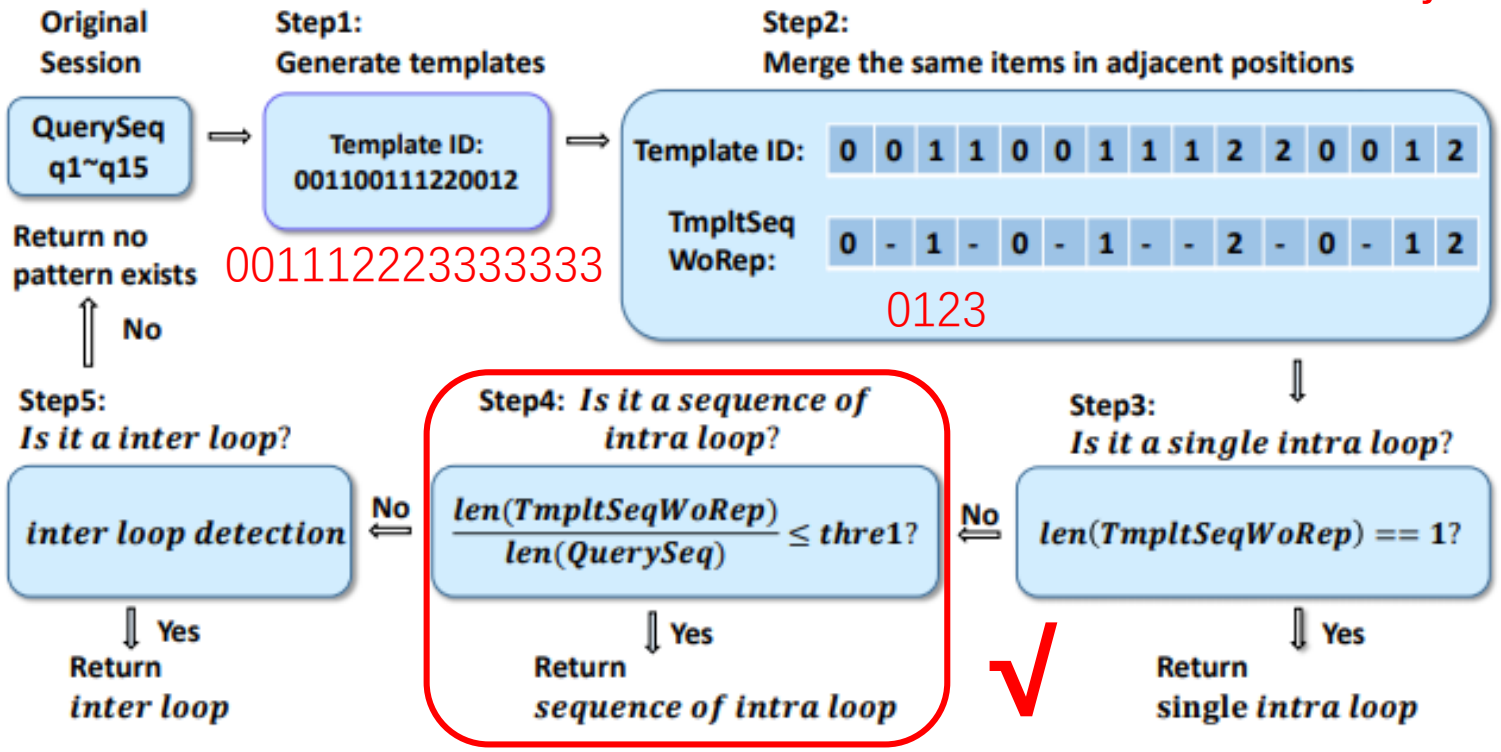


Fig. 2: Overview of loop pattern detection algorithm.

Loop Pattern Detection Algorithm

- Inter loop patterns ([(01⋯)+])

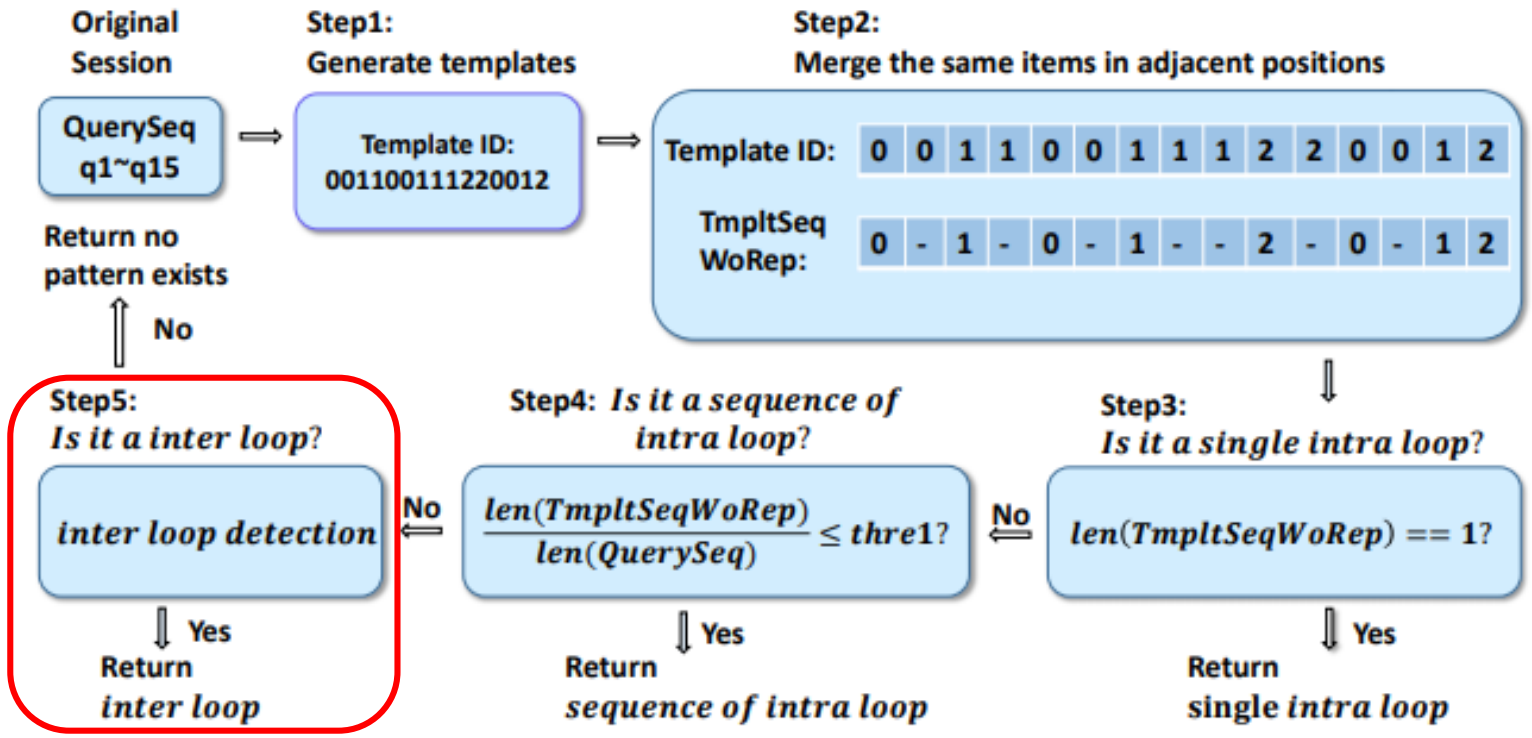


Fig. 2: Overview of loop pattern detection algorithm.

Loop Pattern Detection Algorithm

calculate the maximum subsequence which loops over the entire session.

- Inter loop patterns ([(01⋯)+])

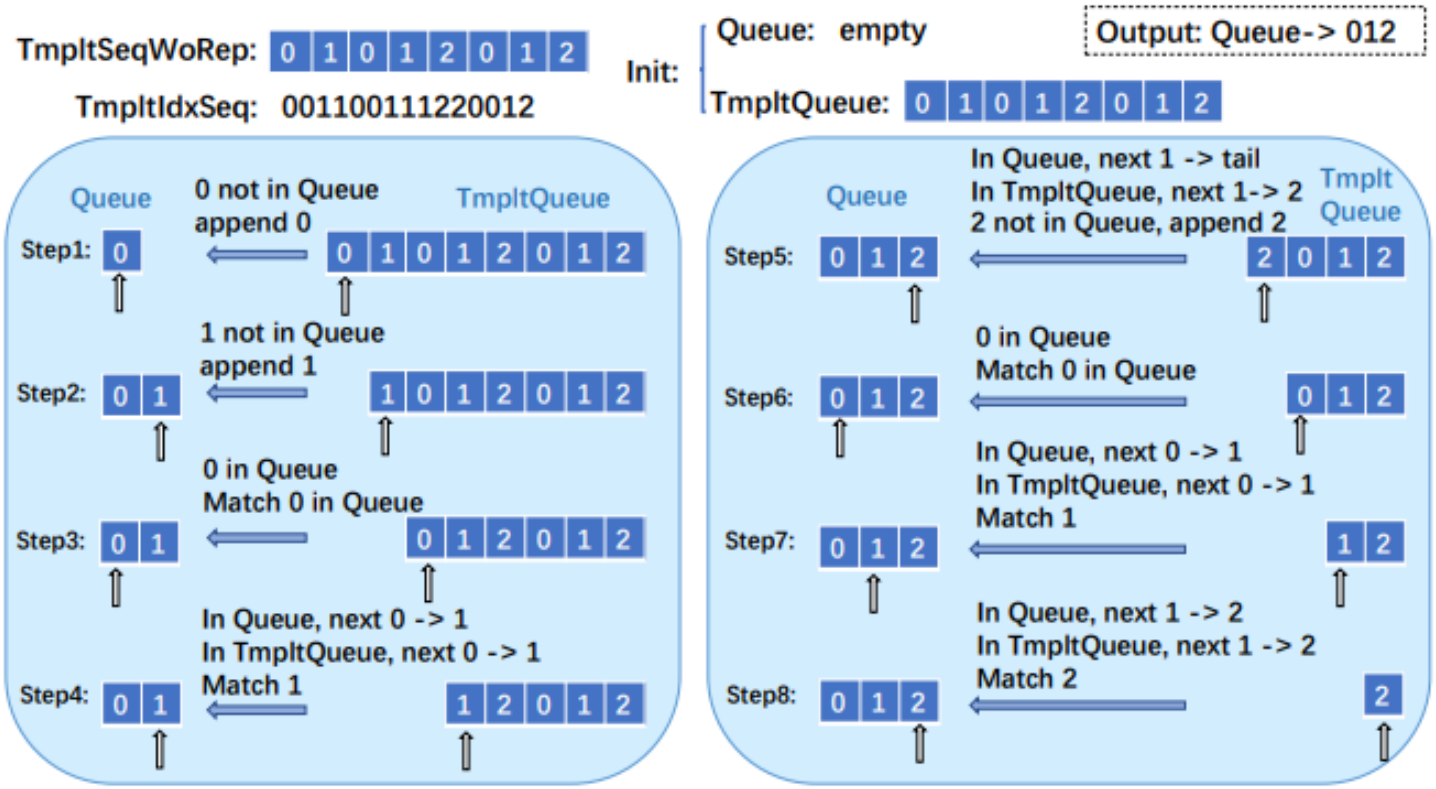


Fig. 3: An example of *inter loop detection*

Loop Pattern Detection Algorithm

- Inter loop patterns ([(01⋯)+])

$$\frac{len(Queue, [012])}{len(QuerySeq, [001100111220012])} <= thre2?$$

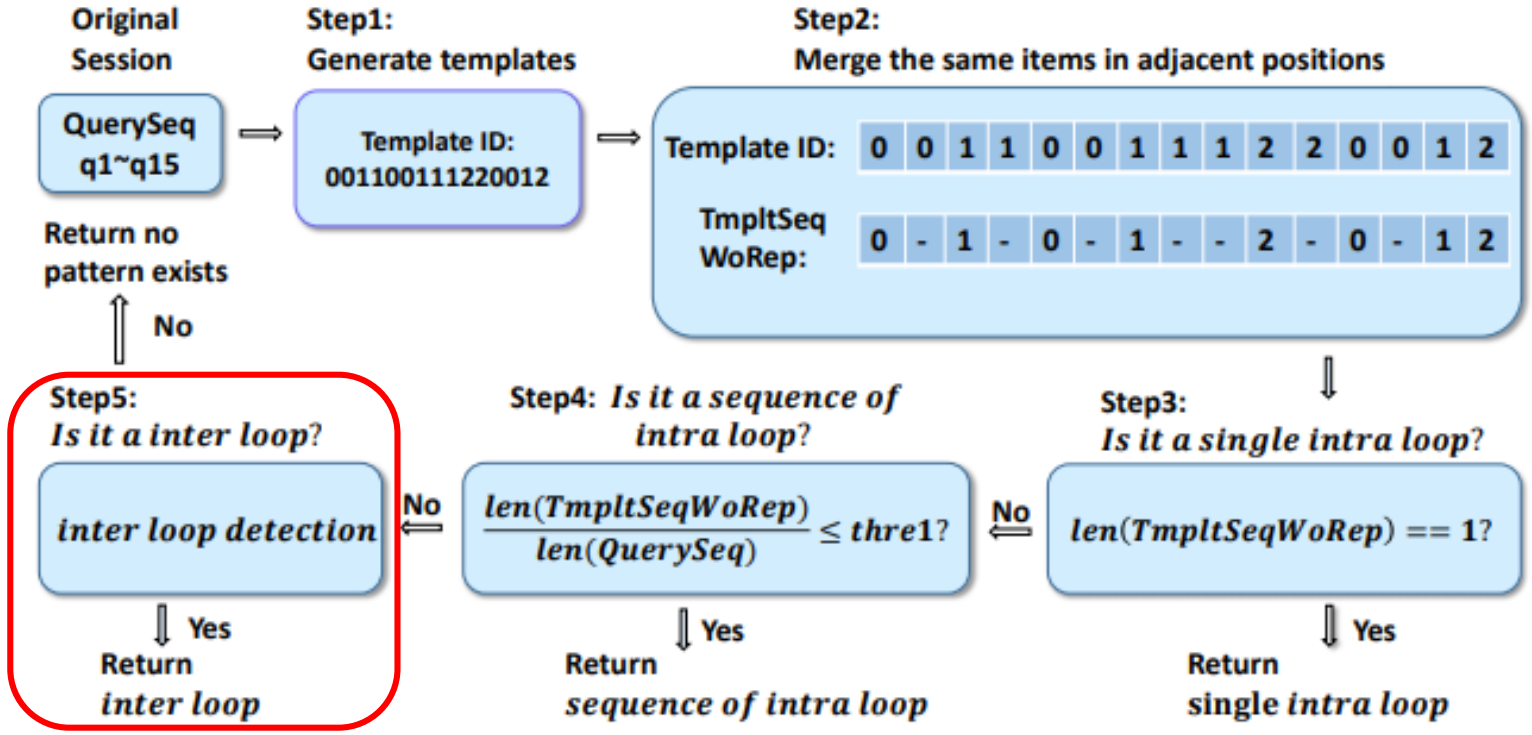
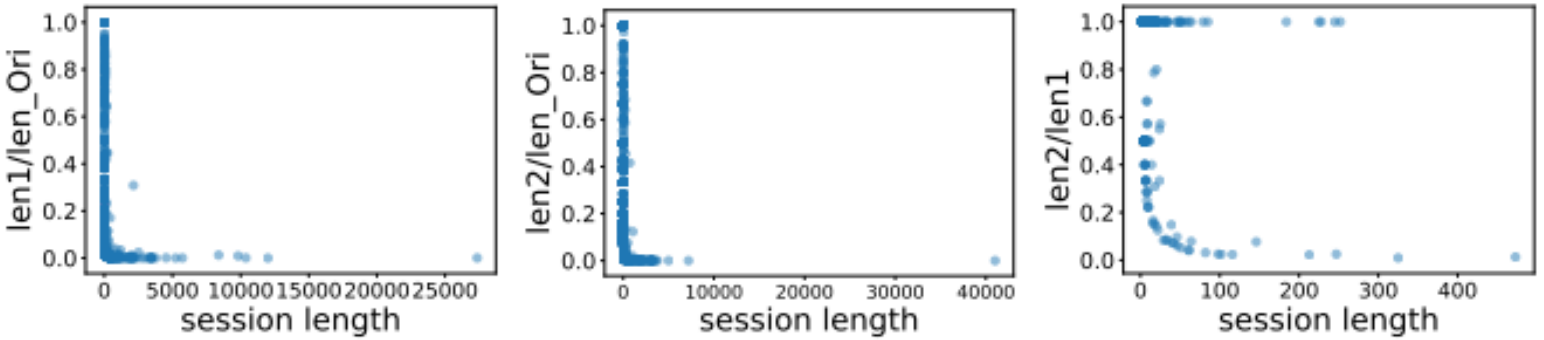


Fig. 2: Overview of loop pattern detection algorithm.

If True, then enough repetition templates exist as inter loop pattern.

Loop Pattern Detection Algorithm

- Threshold Setting



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

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

- len_ori: len(QuerySeq)
- len1: len(TmpItSeqWoRep)
- len2: len(Queue)

- len1/len_ori: present the distribution of intra loop, both single and sequence intra loop pattern.
- len2/len_ori: present the distribution of intra loop and inter loop pattern.
- len2/len1: present the distribution of inter loop

Loop Pattern Detection Algorithm

- Complexity: O(nlogn)

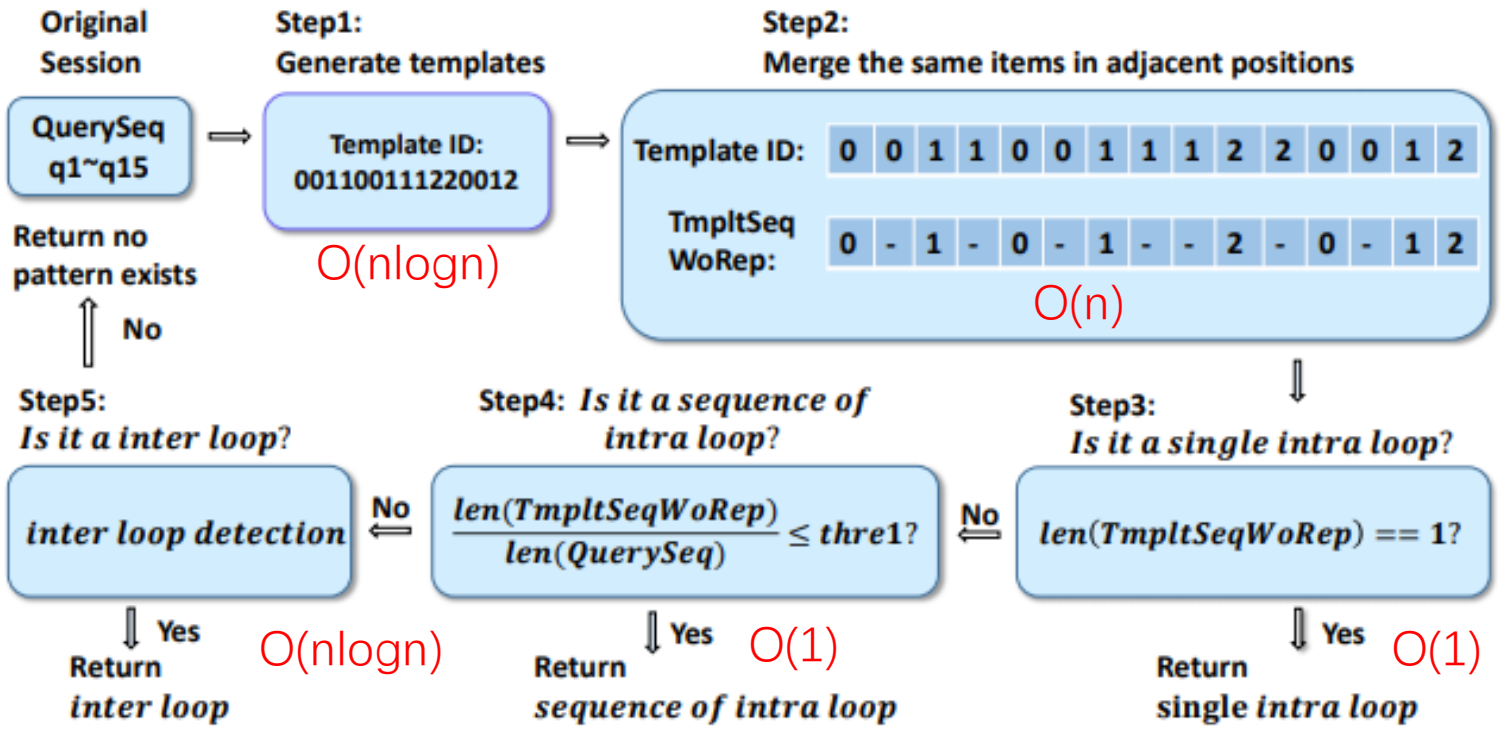


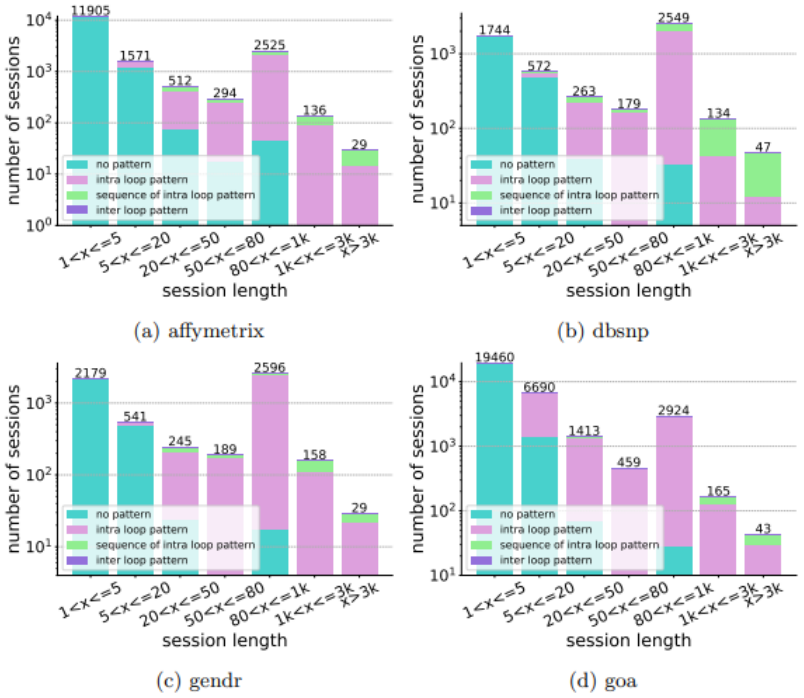
Fig. 2: Overview of loop pattern detection algorithm.

Pipeline Method

1. Frequency Test
2. Loop Pattern Detection Algorithm

Experiments

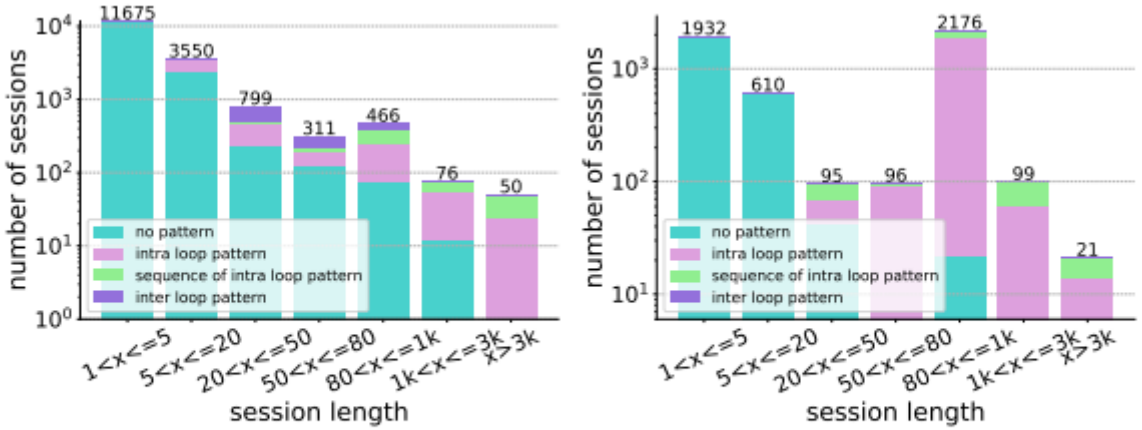
- Loop Pattern Detection Algorithm
  - ➢ our algorithm can recognize all the sessions with lengths more than 1,000, and most sessions with lengths from 80 to 1, 000.



(a) affymetrix (b) dbsnp

(c) gendr (d) goa

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



(a) linkedgeodata (b) linkedspl

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

Experiments

- Robotic and organic query classification pipeline method
  - The distributions for organic queries follow a strong daily rhythm, which indicates a direct human evolvement.
  - For robotic queries, most of them are uniformly distributed.

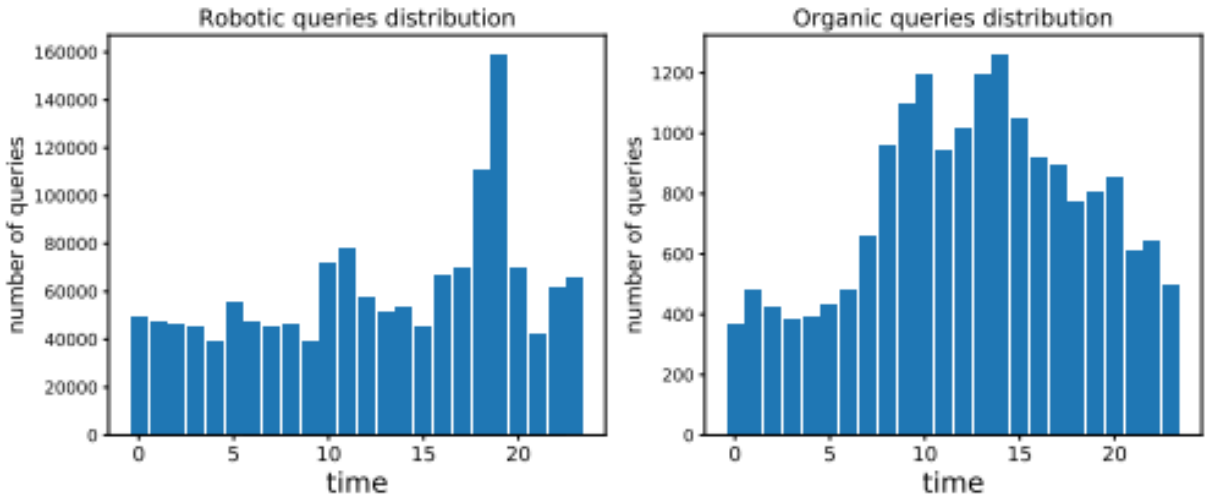


Fig. 7: Query Request Time (UTC) Distribution of the linkedgeodata.

Thanks!