Improving the Quality of the User Experience by Query Answer Modification

João Pedro V. Pinheiro[1], **Marco A. Casanova**[1], **Elisa S. Menendez**[2]

[1] Department of Informatics, PUC-Rio,
22453-900 Rio de Janeiro, RJ, Brazil

[2]Federal Institute of Education, Science and Technology of Sergipe,
49025-330 Aracaju, SE, Brazil

{jpinheiro,casanova}@inf.puc-rio.br, elisa.menendez@academico.ifs.edu.br

Abstract. *This paper proposes a process that modifies the presentation of a query answer to improve the quality of the user's experience. The process is particularly useful when the answer is long and repetitive. The process reorganizes the original query answer by applying heuristics to summarize the results and to select template questions that create a user dialog that guides the presentation of the results.*

1. Introduction

Question Answering (QA) systems combine techniques from multiple fields of computer science, among which: Natural Language Processing (NLP), Information Retrieval, Machine Learning (ML), and Semantic Web. Assuming that the user is interested in querying an RDF knowledge base, a QA system may be split into two parts: *Question*, which receives a user's input in natural language, transforms it into a SPARQL query and searches the RDF knowledge base; and *Answer*, which displays consistent results in a human-readable format to the user.

This paper addresses the problem of *query answer modification* to improve the quality of the user's experience. For example, imagine yourself as a user interacting with a voice virtual assistant, and you ask an open-ended question about a specific subject, e.g., *"Which artists were born on May 30th?"*. The query answer may have a long list of artists, as shown in Table 1. Instead of listing the results, the virtual assistant may formulate questions to the user based on the prior result set, such as: *"Do you want to list American or European artists?"*; *"Do you prefer Jazz, Pop, or Classical music?"*; and *"Do you want to filter by active artists?"*.

The paper proposes a process that reorganizes the original query answer by applying heuristics to summarize the results and to select template questions - explained in detail on Section 3.3 - that create a user dialog that guides the presentation of the results. The heuristics allow deciding which properties returned in the query answer are interesting to apply aggregations (**group by** operations) and which template questions best fit each case. The heuristics also help decide if the answer is ready to be displayed to the user, or if the answer must be improved.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 describes the query answer modification process. Finally, Section 4 presents the conclusions and directions for future research.

Artist	Genre	Birth Date	Death Date	Gender	Nationality
Goodman, Benny	Jazz	1909-05-30	1986-06-13	Male	American
Leonhardt, Gustav	Classical	1928-05-30	2012-01-16	Male	Dutch
Green, CeeLo	Pop	1974-05-30		Male	American
Biosphere	Eletronic	1962-05-30		Male	Norwegian
Fredriksson, Marie	Pop	1958-05-30	2019-12-09	Female	Swedish
Banhart, Devendra	Folk	1981-05-30		Male	American

Table 1. Example of question answer for an open-ended question

2. Brief review of related work

The work in [Dalianis and Hovy 1996] addressed the problem of redundancy in text generation. The authors solved this problem with aggregation, focusing on identifying mechanisms or rules to remove redundant information. As an example, the text *"Yigal is an employee at ISI. Hercules is a visitor at ISI. Eduard is an employee at ISI. Kevin is an employee at ISI. Vibhu is a student at ISI."* can be aggregated as *"At ISI, Yigal, Eduard, and Kevin are all employees; Vibhu is a student; and Hercules is a visitor."*.

The authors developed a questionnaire and applied it to computer scientists. It was composed of five example sets of input data, and each example contained between 11 and 18 propositions. After analysing the answers, four classes or types of aggregation rules became obvious: *Grouping and collapsing rules*, *Ordering rules*, *Casting rules*, *Parsimony rules*. Furthermore, the paper listed eight aggregation strategies, each related to one aggregation class.

As future work, the authors suggested a follow-up study involving a larger group of people, not all of whom being computer scientists for more general results. Also, in the last section, some scenarios presented unsatisfactory results and were listed as future improvements.

In [Deutch et al. 2017], the main idea was an approach to present query results as sentences in Natural Language (NL) with provenance information. The authors argued that the answers in the query result lack justification and suggested the notion of provenance, which corresponds to including additional information to query results. Also, provenance information helps validate answers. The paper used the MAS (Microsoft Academic Research) publication database to validate the results. The proposed solution consisted of the following key contributions: Provenance tracking based on the NL query structure, Factorization, Summarization, and Implementation and Experiments.

An important step was provenance tracking based on the query structure. The authors used two external tools in this process. The modified NaLIR (Natural Language Interface for Relational databases) tool was used to store exactly which parts of an NL query translate to which parts of the formal query. The evaluation of the formal query used the *provenance*-aware engine SelP - Selective tracking and presentation of data *provenance*. It stored which parts of the query "contributed" to which parts of the *provenance*. Thus, two mappings are available for the next steps: **text-to-query-parts** and **query-parts-to-provenance**.

This study was the first to address provenance for the NL queries problem. After

implementation and experiments, the authors listed two main limitations of their work. The sentence generation module was specifically designed to fit NaLIR, and will need to be replaced if a different NL query engine is used. Second, the solution is limited to conjunctive Queries, not supporting unions and aggregations.

Several studies addressed the problem of creating a question-answering (QA) interface to databases. Usually, the proposed QA process has four steps: *Question Analysis*, *Phrase Mapping*, *Disambiguation*, and *Query Construction* - not necessarily in this order. In what follows, we assume that the QA interface is constructed over an RDF knowledge base, accessed through a SPARQL endpoint.

3. The query answer modification process

This section is organized as follows. Section 3.1 describes the process of transforming a single-column into a three-column result set. Section 3.2 addresses the use of frequency analysis based on RDF metadata. Section 3.3 briefly discusses how to construct the user dialog. Section 3.4 details the use of ranking to present the final result set to the user.

3.1. Transforming single-column into three-column result sets

The query answer modification process we propose starts after the query is executed. The expected inputs are the SPARQL query and the result set. There are two possible scenarios: the result set has a *single column*, or the result set has *multiple columns*. Our study focuses on the first case.

We base our discussion on a series of question answering challenges over Linked Data, referred to as QALD - Question Answering over Linked Data[1]. Several papers use QALD to measure quality metrics of system's answers. We noticed that most queries listed in the QALD challenges had single-column answers, which calls for enriching the answers for the purposes of this paper. A simple approach is to add to the instances returned their property values. Indeed, frequently, the answers represent sets of instances of the same *rdf:type*. So, it is straightforward to modify the original SPARQL query to also retrieve the desired property values.

As an example, consider the question *"Which artists were born on May 30th?"*. The result set of the corresponding SPARQL query has instances of type *mo:MusicArtist*, as in Figure 1c. Then, by modifying the original SPARQL query, it is possible to also retrieve property values, as shown in Figure 1d. Note that, in Figure 1d, the column *artist* has repeated values. However, instead of normalizing the returned table, we decided to keep this three-column format to simplify data manipulation.

3.2. Frequency analysis based on computed metadata

Recall that RDF allows representing data and metadata as triples of the form (*s, p, o*), where *s* is the subject, *p* is the predicate and *o* is the object of the triple. Also, recall that an RDF triple set can be interpreted as a directed labeled graph.

In the process we propose, a set of SPARQL queries is used to generate graph statistics, which help decide what to do next. These statistics are related to the frequency of the instances by class and the frequency of the predicates. Subject ranking is based

[1] http://qald.aksw.org

```
prefix mo: <http://purl.org/ontology/mo/>
prefix foaf: <http://xmlns.com/foaf/0.1/>

select distinct ?artist
from <http://musicbrainz.org>
where {
  ?artist a mo:MusicArtist .
  ?artist dbo:birthDate ?date .
  filter(regex(?date, "5-30$", "i")) .
}
```

(a) Single-column query

```
select distinct ?artist ?predicate ?object
where {
  {
    select distinct ?artist
    from <http://musicbrainz.org>
    where {
      ?artist a mo:MusicArtist .
      ?artist dbo:birthDate ?date .
      filter(regex(?date, "5-30$", "i")) .
    }
  } # prior query as subquery
  ?artist ?predicate ?object .
  filter(isLiteral(?object)) .
}
```

(b) Multiple-column query

artist
mo:MusicArtist/1
mo:MusicArtist/2
mo:MusicArtist/3
mo:MusicArtist/4
mo:MusicArtist/5
mo:MusicArtist/6
mo:MusicArtist/7
mo:MusicArtist/8
mo:MusicArtist/9
mo:MusicArtist/10

(c) Single-column result set

artist	predicate	object
mo:MusicArtist/1	foaf:name	"Green, CeeLo"
mo:MusicArtist/1	mo:genre	"pop"
mo:MusicArtist/1	dbo:BirthDate	"1974-05-30"
mo:MusicArtist/1	dbo:DeathDate	""
mo:MusicArtist/1	foaf:gender	"Male"
mo:MusicArtist/1	dbp:nationality	"American"
mo:MusicArtist/2	foaf:name	"Leonhardt, Gustav"
mo:MusicArtist/2	mo:genre	"Classical"
mo:MusicArtist/2	dbo:BirthDate	"1928-05-30"
mo:MusicArtist/2	dbo:DeathDate	"2012-01-16"
mo:MusicArtist/2	foaf:gender	"Male"
mo:MusicArtist/2	dbp:nationality	"Dutch"
mo:MusicArtist/3	foaf:name	"Goodman, Benny"
mo:MusicArtist/3	mo:genre	"Jazz"
mo:MusicArtist/3	dbo:BirthDate	"1909-05-30"
mo:MusicArtist/3	dbo:DeathDate	"1986-06-13"
mo:MusicArtist/3	foaf:gender	"Male"
mo:MusicArtist/3	dbp:nationality	"American"

(d) Multiple-column result set

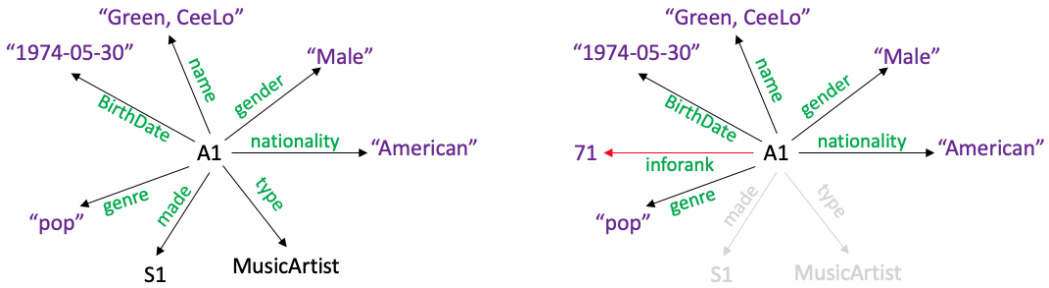
Figure 1. Transformation with SPARQL queries

on *InfoRank*, a family of importance measures proposed in [Menendez et al. 2019]. The *InfoRank* metric helps our process prioritize the most relevant triples of the result set. Its usage is detailed in Section 3.4.

There are two types of frequencies used in the system. A *global frequency* is defined over the full graph and is computed only once before any query is executed. On the other hand, a *local frequency* is defined over the sub-graph generated as in Section 3.1 and is computed at run time. It is important to highlight that both *global* and *local frequencies* are computed over predicates pointing to literals only.

As an example, Figure 2 shows an instance **A1**. The initial state (Figure 2a) has predicates pointing to literals and other instances. Notice that the final state (Figure 2b) only has predicates pointing to literals and an extra predicate called *inforank*.

A parameterized threshold is used to filter predicates that are candidates to be used in a **group_by** operation. By default, this threshold values is set to 0.4, which means the predicate must appear in at least 40% of the unique subjects.



(a) All predicates (b) Filtered predicates

Figure 2. Filtered predicates by literal and highlighted *InfoRank*

3.3. Computing the user dialog

As mentioned in the previous section, an aggregation process is applied over the filtered predicates. These predicates are evaluated sorted by its *local frequency*. Another threshold is related to the number of aggregated values. If this number is less than or equal to the default value - which is 5 - then the process chooses which template question must be used and returns a single new question to the user.

These template questions are choice questions, formulated in natural language, that offer aggregated predicates as alternatives to the user. Using the same example as before, the filtered predicates are: *foaf:gender*, with two aggregated values - *female* and *male*; and *dbo:background*, with three aggregated values - *non_performing_personnel*, *non_vocal_instrumentalist*, and *solo_singer*. Thus, the process may generate the following questions :

1. Do you prefer {*female*}, or {*male*} artists?
2. Between {*non_performing_personnel*}, {*non_vocal_instrumentalist*}, and {*solo_singer*}, which artists do you prefer?

Since *foaf:gender* precedes *dbo:backgroud* in the computed frequencies, only the first question displayed above is returned to the user. If the user keeps interacting with the system, the whole process restarts.

3.4. Ranking the final results

This last step is only triggered when the process detects that there are no other candidate predicates for aggregation. Thus, a result set must be submitted to the user. The *inforank* metric is used to sort and filter relevant information in the final answer.

Finally, since one of the main goals of the query answer modification process is to reduce the final query answer, the number of rows returned may be limited.

4. Conclusions

This paper proposed a process that reorganizes a query answer to improve the user's experience. The process is based on heuristics to summarize the results and to select template questions that create a user dialog that guides the presentation of the results. The heuristics allow deciding which properties returned in the query answer are interesting to apply aggregations (**group_by** operations) and which template questions best fit in each case.

Parameterized thresholds allow customized behavior for each scenario. These thresholds are used to filter interesting properties to apply aggregations and to select template questions that best fit in each case. Also, this approach is easily pluggable into any QA system.

Future work will contemplate further experiments to improve the heuristics that drive the user dialog based on template questions.

References

Dalianis, H. and Hovy, E. (1996). Aggregation in natural language generation. In Carbonell, J. G., Siekmann, J., Goos, G., Hartmanis, J., Leeuwen, J., Adorni, G., and Zock, M., editors, *Trends in Natural Language Generation An Artificial Intelligence Perspective*, volume 1036, pages 88–105. Springer Berlin Heidelberg, Berlin, Heidelberg.

Deutch, D., Frost, N., and Gilad, A. (2017). Provenance for natural language queries. *Proceedings of the VLDB Endowment*, 10(5):577–588.

Diefenbach, D., Lopez, V., Singh, K., and Maret, P. (2018). Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information Systems*, 55(3):529–569.

Hu, X., Dang, D., Yao, Y., and Ye, L. (2018). Natural language aggregate query over RDF data. *Information Sciences*, 454-455:363–381.

Isotani, S. and Bittencourt, I. I. (2015). *Dados abertos conectados*. Novatec, São Paulo. OCLC: 959729676.

Menendez, E. S., Casanova, M. A., Leme, L. A. P., and Boughanem, M. (2019). Novel node importance measures to improve keyword search over rdf graphs. In *International Conference on Database and Expert Systems Applications*, pages 143–158. Springer.

Pan, J. Z., Vetere, G., Gomez-Perez, J. M., and Wu, H., editors (2017). *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Springer International Publishing, Cham.

Reutter, J. L., Soto, A., and Vrgoč, D. (2015). Recursion in SPARQL. In Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Thirunarayan, K., and Staab, S., editors, *The Semantic Web - ISWC 2015*, volume 9366, pages 19–35. Springer International Publishing, Cham.