Proceedings of the 8th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'22)
Prague, Czech Republic - July 28- 30, 2022
Paper No. CIST 118
DOI: 10.11159/cist22.118

Using Domain Knowledge to Rank SPARQL Query Results According to User Preferences

Neli Zlatareva
Central Connecticut State University
1615 Stanley Street, New Britain, CT 06050, USA
zlatareva@ccsu.edu

Abstract – A growing number of applications for the Semantic Web are expected to be able to address user preferences in addition to well defined constraints when searching through large volumes of linked data. SPARQL, the most popular language for querying linked data repositories, does not have built-in tools for representing and processing preferences. The need for more adequate expression of user queries is well recognized by the Semantic Web community and two main approaches were offered to address this need. The *quantitative* approach associates a scoring function with each query result allowing results to be ordered according to their relevance to user preferences, while the *qualitative* approach relies on extending SPARQL with modifiers to express and rank preferences. The major concern with the former is that numerical scoring functions are hard to define, while the later cannot handle multiple preferences at the same time. In this paper, we suggest utilizing domain knowledge to process and rank SPARQL query results according to user preferences without the need to modify SPARQL functionality. An example application is used throughout the paper to illustrate the proposed approach.

***Keywords*:** SPARQL, Semantic Web, Linked Data, RDF

1. Introduction

SPARQL is the most popular language for querying linked data repositories as well as the current recommendation of the W3C (https://www.w3.org/TR/sparql11-query/). It allows for different query forms depending on user needs, namely SELECT, CONSTRUCT, DESCRIBE, and ASK. The most widely used query form, the SELECT query, uses a pattern-matching algorithm to retrieve information from RDF datasets, while DESCRIBE and CONSTRUCT queries return RDF datasets according to a query template. The ASK query returns a Boolean value depending on whether a query pattern has any solution or not. Although SPARQL is a powerful query language, its pattern matching capability is limited to a simple entailment which directly matches patterns to RDF triples.

Depending on how the SELECT query is stated, the list of query results may be unnecessary long or completely empty. To handle a large number of query results, SPARQL provides solution modifiers (ORDER BY, LIMIT, DISTINCT, etc.) to trim/order the results, as well as the FILTER expression to filter the results according to a given Boolean expression. In case of an empty query result, SELECT query can be relaxed by utilizing modifiers OPTIONAL and UNION. Although these built-in capabilities allow for creating flexible SELECT queries, they are not always capable to adequately capture user preferences and thus provide a proper ordering of query results. Consider, for example, an application helping a tourist to find a restaurant of their choice. Not having specific information about the area, the query will probably be defined as a set of requirements and preferences. Integrating a single preference (for example, "I prefer a Mexican restaurant") is easy by FILTERing only "Mexican" restaurants. However, if the user prefers an affordable Mexican restaurant at a walking distance, then FILTERing all Mexican restaurants, followed by FILTERing all "affordable" ones, and finally FILTERing those "at a walking distance" may produce an empty result. The problem is that user preferences are treated by SPARQL as constraints similar to all other query patterns.

The need to represent and process preferences in SPARQL queries is well recognized by the Semantic Web community [1 - 3]. Efforts to address this need fell in one of the two approaches. The *quantitative* approach associates a scoring function with each query result allowing results to be ordered according to their "relevance" to user preferences. An extension of SPARQL, f-SPARQL [4], implements this idea by utilizing fuzzy predicates to express preferences such as *close*, *recent*, etc. Such predicates are represented by a membership function and processed by an extended FILTER expression, *FILTER (?X <op> <fuzzy term>) [WITH DB]*, where *<op>* is a Boolean operator and *DB* is a degree of belief in the range {0, 1}. For example, FILTER (?distance = walking) with 0.7, where 0.7 is a measure stating the importance of this criterion in the

final ordering of query results. Such expressions are processed by converting them into Boolean expressions which are handled by SPARQL in a conventional way. The major concern with this approach is that numerical scoring functions are expected to be defined by the user, which might be difficult or even impossible. For example, if the user is looking for a restaurant at a walking distance with at least 4-stars rating, the scoring function must reflect the normalized weighted average of both preferences, the distance, and the rating.

The alternative, qualitative, approach allows the user to express their preferences directly without the need to quantify them. [5] suggests a new solution sequence modifier, PREFERRING, allowing to express user preferences as part of the query itself. Similar idea is proposed in [6] where a version of SPARQL called PrefSPARQL is introduced. In both cases, preferences are addressed by the form SELECT ?X WHERE {P1 PREFERRING P2}. [7] presents a framework called Morph-Skyline++ which processes SPARQL qualitative preferences by converting them into SQL queries to be evaluated by a relational database management system. One of the limitations of the qualitative approach is that it does not allow multiple preferences to be handled at the same time.

In this paper, we suggest using domain knowledge to handle the final ordering of query results according to user preferences without the need to modify SPARQL functionality. We assume that the linked data repository to be searched contains deductively closed datasets relevant to the user's query. To present and illustrate our approach, we use an example application presented next.

2. Motivation example and outline of the proposed framework

Consider the "restaurant search" scenario and assume that the user is looking for a restaurant at a walking distance from their current location which is not very busy so that they can be served reasonably fast. Also, the user prefers a place serving burgers or tacos and is offering 4- or 5-stars non-vegetarian specials at a reasonable price.

Assume that the information about local infrastructure (restaurants, hotels, etc.) is available on a linked data network (the experiments were conducted on a small, simulated network). It is not to be expected from the user to state their query in SPARQL, but a friendly interface helping the user to describe the search request in a semi-formal structured way can help simplify the translation of the request into a SPARQL query. Notice that the informal query suggests two different types of expectations, *constraints* and *preferences*. The user can be asked to state them separately as follows: i.) state the search object (a restaurant; this will limit the scope of the search) and define its desired characteristics (walking distance from the current location, reasonable wait times, etc.), and ii.) state additional preferences (restaurant/food choices, ratings, specials, price, etc.). Converting the resulting "query pieces" into a SPARQL query is a huge task on its own which is addressed by a growing number of research in the area of Question-Answering systems [8, 9] including our own [10]. In this paper, we assume that the user request is already converted into a SPARQL query, and the emphasis is on the processing and ranking of query results according to the explicitly stated user preferences. Because preferences cannot be expressed directly in SPARQL, the generated query treats all preferences as constraints which may or may not be satisfied. This suggests a potentially long list of results that are not amenable to any ordering because user preferences are not ordered.

Here is the SPARQL query capturing our example scenario:

```
SELECT DISTINCT ?food ?protein ?rating ?special ?price ?place ?type
WHERE {
   {SELECT DISTINCT ?place ?type ?busyness
    WHERE {?place rdf:type :Restaurant .
            ?place :hasType ?type .
       Optional  {?place :hasBusyness ?busyness . }
       Optional  {?place :hasDistance ?distance . }
       Filter not exists {?place :hasReservation true}
       Filter (?distance != :DrivingDistance || !bound(?distance))
       Filter exists {{?place :hasType 'American'} UNION {?place :hasType 'Mexican'}}}}
  {{?place :hasType 'Mexican' .
  ?food rdf:type taco:Taco .
  ?food taco:hasPrice ?price .
  ?food taco:hasProtein ?protein .
```

```
Optional {?food taco:hasRating ?rating . }
Optional {?food taco:daySpecial ?special .}
Filter (?rating >= "4"^^xsd:integer || !bound(?rating))}
UNION
{?place :hasType 'American' .
?food rdf:type :Hamburger .
?food :hasPrice ?price .
?food :hasProtein ?protein .
Optional {?food :hasRating ?rating . }
Optional {?food :daySpecial ?special .}  }
Filter (?rating >= "4"^^xsd:integer || !bound(?rating))}}
```

This query returns the following result list shown informally for brevity:

food	protein	rating	special	price	place	type
MyOwnCheeseBurger	Beef	5		25	BestTavern	American
BlackBeanQuinoaBurger	Veggie	5	true	15	BestTavern	American
ChutegateCheeseBurger	Turkey			12	BestTavern	American
KidsCheeseBurger	Beef	4	true	9	BestTavern	American
MyOwnKidsBurger	Beef	5		12	BestTavern	American
BlackBeanBurger	Veggie	4	true	15	BestTavern	American
ChutegateSpicyCowboyBaconBurger	Beef	5	true	22	BestTavern	American
ChutegateBaconCheeseBurger	Beef	4	true	18	BestTavern	American
SpicyLoversTaco	Chicken	4	true	14	BestTacos	Mexican
SpicyLoversTaco	Beef	4	true	14	BestTacos	Mexican
VeggieOverloadTaco	Veggie	5		15	BestTacos	Mexican
MeatLoversTaco	Chicken	5	true	16	BestTacos	Mexican
MeatLoversTaco	Beef	5	true	16	BestTacos	Mexican
StuffedTaco	Chicken	4		12	BestTacos	Mexican
NoMeatNoProblemTaco	Veggie	5	true	11	BestTacos	Mexican

The identified two restaurants satisfy explicitly defined user constraints and preferences. The remaining attributes of the results, however, cannot be used for further ranking of the choices, because no explicit ordering of user preferences is stated. To further process and order these results to provide the user with a small set of best choices we can use the underlying domain ontology as described next.

Recall that the informal user query contains references to "non-vegetarian" food which the underlying ontology identifies as a class with members {Beef, Turkey, Chicken, SeaFood}. Similarly, the remaining constraints and preferences associated with ratings, specials, prices, etc. are classified as instances of the respective classes. This input is processed by a *rule pattern extractor* module which returns the so-called *master rule* defining the context where each query result is to be evaluated. The master rule consists of two lists, where each list in turn contains sets of values, and a conclusion pattern. It has the following general form:

$(\{\text{individual/value}_1,\ldots, \text{individual/value}_n\}, \ldots, \{ \text{individual/value}_1,\ldots, \text{individual/value}_n\})$
$(\{\text{individual/value}_1,\ldots, \text{individual/value}_n\}, \ldots, \{ \text{individual/value}_1,\ldots, \text{individual/value}_n\}) \rightarrow$
$\rightarrow (\text{individual/value}_1,\ldots, \text{individual/value}_n)$

The first list enumerates best choices matching either a constraint or a preference, while the second list contains acceptable choices of user preferences. The conclusion specifies the format of the recommendations to be returned to the user. It is assumed that all constraints are satisfied and thus they do not have to be included in the master rule. However, if a constraint is referenced in the rule's conclusion, it must be added to the first list to allow for proper unification with the recommendation choice.

For our example, the master rule looks as follows:

({Beef, Turkey, Chicken, SeaFood}, {5}, {true}, {MIN}, {American, Mexican})
({Veggie}, {4}, {false, NONE}, {ANY}]) → (place, food)

To match query results to the master rule, each query result is converted to a rule with the same structure and semantics as the master rule. Here are the rules representing three "best" choices:

({Beef}, {5}, {true}, {American}) ({22}) → (BestTavern, ChutegateSpicyCowboyBaconBurger)

({Chicken, Beef}, {5}, {true}, {Mexican}) ({16}) → (BestTacos, MeatLoversTaco)

({5}, {true}, {Mexican}, {11}) ({Veggie}) → (BestTacos, NoMeatNoProblemTaco)

Each on of these choices "dominates" the other choices by the number of satisfied preferences contained in the first list. Each has three satisfied preferences in the first list and 1 acceptable preference in the second list, while all other choices have a smaller number of satisfied preferences. Among the three best choices, it is up to the user to decide whether they will trade the food choice for the price.

The advantage of the proposed framework is that there is no need for the user to "rank" their preferences, which may not even be possible especially at the time the query is posted. Query generation relies on standard SPARQL functionality, while the final evaluation and ranking of query results is left to a post-query processing module utilizing the domain ontology.

3. Conclusion

The paper discussed how domain knowledge can be used to process and rank SPARQL query results according to user preferences without the need to modify SPARQL functionality. An example application was used to illustrate the proposed approach.

Acknowledgement. This research was partially supported by a CCSU-AAUP research grant.

References

[1] Patel-Schneider P.F., Polleres A., Martin D. (2018) Comparative Preferences in SPARQL. In: Faron Zucker C., Ghidini C., Napoli A., Toussaint Y. (eds) Knowledge Engineering and Knowledge Management. EKAW 2018. Lecture Notes in Computer Science, vol 11313. Springer, Cham. https://doi.org/10.1007/978-3-030-03667-6_19

[2] Olivier Pivert, Olfa Slama, Virginie Thion. (2016) SPARQL Extensions with Preferences: a Survey. ACM Symposium on Applied Computing, Pisa, Italy, France. 10.1145/2851613.2851690. hal-01235190

[3] Ayala, V., Przyjaciel-Zablocki M., Hornung T., Schatzle A., Lausen, G. (2014) Extending SPARQL for Recommendations. In *Proceedings of Semantic Web Information Management on Semantic Web Information Management - SWIM'14*.

[4] Cheng, J., Ma, Z., and Yan, L. (2010). f-SPARQL: a flexible extension of SPARQL. In *Proc. of DEXA*, pages 487–494.

[5] Siberski, W., Pan, J. Z., and Thaden, U. (2006). Querying the Semantic Web with Preferences. In *Proc. of ISWC*, pages 612–624.

[6] Gueroussova, M., Polleres, A., and McIlraith, S. A. (2013). SPARQL with qualitative and quantitative preferences. In *Proceedings Intl. Workshop OrdRing*, co-located with ISWC, pages 2–8.

[7] Goncalves, M., Chaves-Fraga, D., and Corcho, O. (2021) Handling Qualitative Preferences in SPARQL over Virtual Ontology-Based Data Access, Semantic Web Journal, IOS Press.

[8] Trivedi, Priyansh and Maheshwari, Gaurav and Dubey, Mohnish and Lehmann, Jens (2017) "Lcquad: A corpus for complex question answering over knowledge graphs" in International Semantic Web Conference, pages 210--218.

[9] Abujabal A., M. Yahya, M. Riedewald, and G. Weikum (2017) - Automated template generation for question answering over knowledge graphs. *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee.

[10] Zlatareva, N. and Amin D. (2021). Natural Language to SPARQL Query Builder for Semantic Web Applications. *Journal of Machine Intelligence and Data Science*, Vol. 2, Avestia Pibl.