Demonstrating Interactive SPARQL Formulation through Positive and Negative Examples and Feedback

Akritas Akritidis and Yannis Tzitzikas
Institute of Computer Science, FORTH-ICS, Greece, and
Computer Science Department, University of Crete, Greece
akritas@csd.uoc.gr,tzitzik@ics.forth.gr

ABSTRACT

The formulation of structured queries in Knowledge Graphs is a challenging task since it presupposes familiarity with the syntax of the query language and the contents of the knowledge graph. To alleviate this problem, for enabling plain users to formulate SPARQL queries, and advanced users to formulate queries with less effort, in this paper we introduce a novel method for "SPARQL by Example". According to this method the user points to positive/negative entities, the system computes one query that describes these entities, and then the user refines the query interactively by providing positive/negative feedback on entities and suggested constraints. We shall demonstrate SPARQL-QBE, a tool that implements this approach, and we will briefly refer to the results of a task-based evaluation with users that provided positive evidence about the usability of the approach.

KEYWORDS

Interactive Query Formulation, SPARQL, Knowledge Graphs

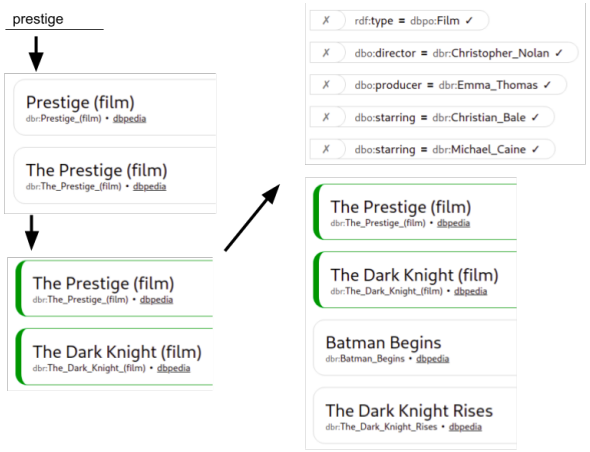

Figure 1: Finding movies through examples and feedback (via SPARQL-QBE)

1 MOTIVATION AND NOVELTY

To enable plain users to formulate SPARQL queries, and aid knowledgeable users to formulate SPARQL queries with less effort, in this paper we present "SPARQL by Example", a novel interactive method for formulating queries. The approach is inspired by the *Query-by-Example* paradigm [5] that was developed in the context of Relational Databases, as well as by the *relevance feedback* mechanisms in Information Retrieval.

Although there are some works that aim at offering a QBE-like interaction over Knowledge Graphs in RDF, i.e. [1, 3, 4], our approach has some distinctive characteristics. In particular, "Qbees" [4] neither supports negative examples nor produces a pure SPARQL query, "Query from examples" [3] requires from the user to answer a number of questions, while "Reverse engineering SPARQL queries" [1] cannot receive feedback on the generated constraints.

2 APPROACH

The main idea is the following: The user provides one or more entities, that (s)he may have discovered while browsing or by keyword search. We then compute one query whose result contains the provided plus other entities that have commonalities with the entities provided by the user. Then the user can refine the formulated answer (and query) by providing interactively positive/negative feedback, by selecting/rejecting constraints that are given to the user, as well as positive/negative examples.

2.1 The tool SPARQL-QBE in Brief

To grasp the idea, Figure 1 shows the start of the interaction loop. The user selects (through keyword search) as starting examples the films "The Prestige" and "The Dark Knight". The system then generates the list of common constraints and returns all entities that comply with them. The user then has the option to either access the results and the generated SPARQL query, or continue refining the results by giving feedback. In our case the user by typing the titles of two movies, managed to formulate the query "movies with Christian Bale and Michael Caine, with director Christoopher Nolan and producer Emma Thomas, and can directly see the answer of that query (the movies "Batman Begins" and The "Dark Knight Rises" as shown in Figure 1(right)), as well as the SPARQL query, i.e.:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT ?x WHERE {
  ?x rdf:type dbo:Film .
  ?x dbo:director dbr:Christopher_Nolan .
  ?x dbo:producer dbr:Emma_Thomas .
  ?x dbo:starring dbr:Christian_Bale .
  ?x dbo:starring dbr:Michael_Caine .
}
```

If the user is not interested in the involvement of the actor "Christian Bale" the constraint "dbp:starring = Christian Bale" can be flagged as unwanted. The system will then generate a new list of constraints that still describe all the examples but ignore the actor.

2.2 The Interactive Loop Algorithm

The process starts by selecting (by browsing, keyword search or any other access method) one or more positive examples E_P. Then it enters into an interaction loop where the system can accept four kinds of input: the positive E_P and negative E_N set of entities, and the positive C_P and negative C_N set of constraints. Note that constraints over path expressions are also supported. Given a dataset and the 4 inputs, the system will output always the same query q (i.e. the order of feedback does not affect the output), and the answer of the query A. The steps of the above process are shown in Alg. 1. In particular, in each loop of the interaction, after inspecting the answer A of the current query, the user can either: (i) provide positive or negative feedback to the entities of the answer received by selecting elements of A, defining in this way the new set of positive and negative examples $E'_P = (E_P \cup E_{P_{added}}) \setminus E_{P_{deleted}}$, and $E'_N = (E_N \cup E_{N_{added}}) \setminus E_{N_{deleted}}$), (ii) delete one of the constraints c of the original query q, and this changes the set of unwanted constraints, i.e. $C_{N'} = C_N \cup \{c\}$, (iii) add a new constraint by clicking on the $(p, =, v)$ values of the elements of the answer; he can also add such a constraint in negated form $(p \neq v)$, and get $C_{P'} = C_P \cup \{(p, op, v)\}$.

Line 2 of the algorithm finds the *common constraints* of the selected entities, defined as $CC(E_P) = \{ (p, v) \mid \forall e \in E_P, (e, p, v) \in KG\}$ where KG is the underlying knowledge graph. Line 3 computes those queries of the powerset of C whose answer does not contain any of the negative examples. As regards ranking (line 4) the elements of Q are ranked according to their expected answer size, for providing a query relaxation behaviour. A smaller answer size is considered as a more constrained query that fits all the requirements. The size of each query in Q can be approximated by the frequency of its constraints (method M_{freq}) or evaluated and computed exactly (method M_{conj}). The latter is more expensive and it is suggested if the size of the KG is not very big.

Algorithm 1 QBE Interactive Loop

1: **function QBE-Interactive**(E_P, E_N: sets of entities, C_P, C_N: sets of constraints)
2: $C \leftarrow (CC(E_P) \cup C_P) \setminus C_N$
3: $Q \leftarrow \{q \mid q \in P(C), ans(q) \cap E_N = \emptyset\}$
4: $q \leftarrow Rank(Q)[1]$ ▷ the first query in the rank
5: **while** $ans(q) = E_P$ **do** ▷ no extra entities
6: $q \leftarrow$ next element of $Rank(Q)$ ▷ The next in the rank query
7: **end while**
8: show $A = and(q)$
9: Receive any input from the user and get E'_P, E'_N, C'_P, C'_N
10: call **QBE-Interactive** (E'_P, E'_N, C'_P, C'_N)

2.3 Demonstration Scenario

The application starts in a keyword search mode for enabling the user to find the starting examples, an example is shown in Figure 2. The user can continue in this way for providing more examples, as shown in Figure 2 (bottom part).

After the user selects any number of examples and exits the keyword search mode, i.e. it presses "Query Formulation" at the top bar, the application provides the first list of constraints and the corresponding results. The user can delete unwanted constraints, can provide more positive/negative examples, as shown in Figure 3.

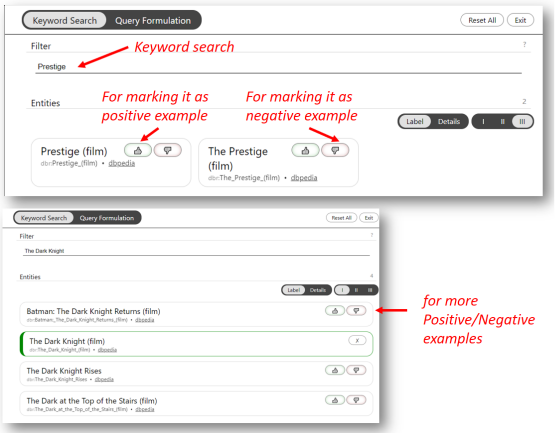

Figure 2: SPARQL-QBE**: Step 1: Keyword Search and marking positive and negative examples**

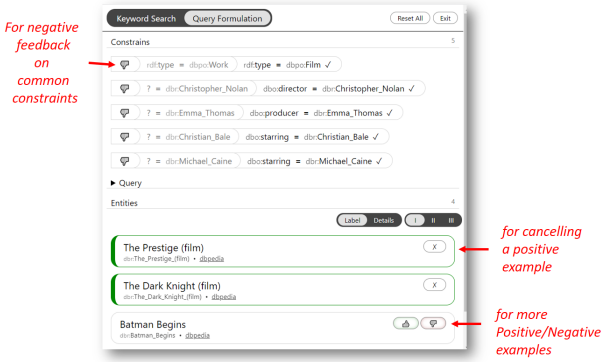

Figure 3: SPARQL-QBE**: Step 2: Negative feedback on common constraints, and more positive/negative examples**

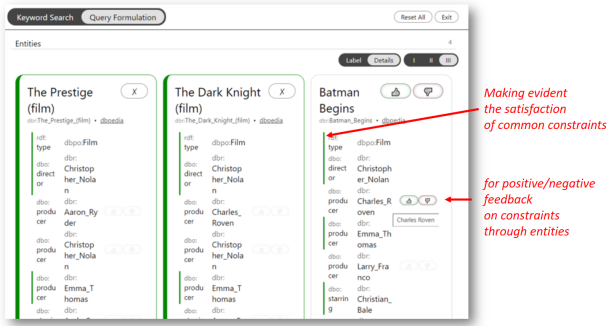

Figure 4: SPARQL-QBE**: Making evident constraint satisfaction, and constraint feedback through entities**

Moreover the tool makes evident the constraints that each entity satisfies (marked green), and enables positive/negative constraints through the entities, as shown in Figure 4.

For the support of property paths, every property value can be extended to display all of its property-value pairs in an indented second list as shown in Figure 5. The user can then create a property path constraint by selecting one of the second list property value pairs. This behaviour is supported recursively, i.e. the user can extend a property value from the second list into a indented third list.

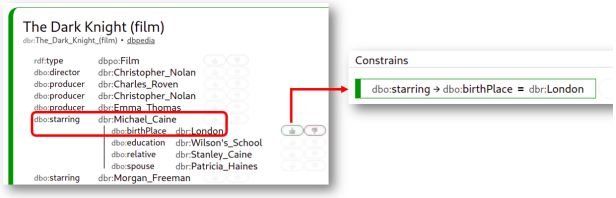

Figure 5: Feedback with path expressions

2.4 Expressive Power

By interacting with the system the system eventually formulates one conjunctive query. Given that the process starts by giving one or more positive examples, these examples should belong to the same class, or have a superclass in common. If the examples belong to different classes then it is not clear how such input should be interpreted.

Except from class-based restriction, the rest constraints of the generated query are positive or negative property-value matches. Since the system allows path expressions, the query can contain various path-based restrictions, consequently, even if all the examples are of the same (direct or indirect) class, the property paths indirectly allow for constraints that relate to multiple other classes.

3 COMPARISON WITH RELATED APPROACHES AND SYSTEMS

There are several tools, "example"-based or not, that can generate queries like those that can be formulated by SPARQL-QBE. However the process that the user has to follow is fundamentally different.

Comparison with systems that do not support the notion of "Example". Without examples, the first step a user has to make is to provide/select a property, a value of that property, or in general some constraint on that property. Even if a system provides a list of all available properties and values of the KG, the user still has to know the commonalities of the desired entities and how they are represented in the context of the tool used. Instead, with the notion of example, the only requirement for the user is to provide a single example (two examples are suggested but not required), and then the feedback loop will assist him/her to generate the desired query.

Consequently, we could say that in general we have two possible starting points: (a) single property constraints (for not example-based systems), and (b) single examples (for example-based systems). The expressive power (or restriction capability) of the two options are not equal. For instance, from two examples a system may infer several (common) constraints, but from two property constraints a system cannot infer anything more. Therefore, in cases where the desired number of constraints is large, even if these constraints are known, an example-based approach may be faster than non example-based systems (like Faceted Search). We should also note that in SPARQL-QBE, apart from the support of examples, if the user cannot provide any example, (s)he can alternatively start by providing a wanted constraint and then continue with the feedback loop; this is a distinctive feature of SPARQL-QBE in comparison to other example-based systems.

Comparison with systems that support the notion of "Example". Compared to example-based systems like "Query from examples" [3], instead of explicitly asking for more information from the user before a query is provided to the user, SPARQL-QBE provides the best query and then through the feedback loop the user can provide more information of his own choosing. In comparison to "Reverse engineering SPARQL queries" [1], that work cannot receive feedback on the generated constraints, therefore has lower interactivity.

4 IMPLEMENTATION AND EVALUATION.

The implementation of SPARQL-QBE is a JavaScript application, with no need for a server for the time being. All the triples are contained in a JSON file packaged with the application that the user loads once with the first load of the application.

4.1 Datasets

We have tested various datasets. For the needs of the task-based evaluation with users we selected a dataset containing most well documented films and actors from DBpedia whose size is 1,083,029 triples (112,668 films, 43,157 actors). A deployment of SPARQL-QBE with this dataset is accessible through https://demos.isl.ics.forth.gr/SPARQL-QBE/.

4.2 Efficiency

Overall, for the datasets described earlier, we have real time interaction. At the initialization of the system the more time consuming task is the downloading of the dataset from the server to the browser of the client (for the compressed movie dataset: 6.4MB ~ 2.5s). Below we report execution times assuming a dataset with one 1 million of RDF triplets. The two main operations are the "Keyword search" with average time 126ms and the "Query execution" with average time 103ms. For the two ranking methods M_{conj} and M_{freq} with caching we achieve executions under 10ms, however for the first time their execution is equivalent to a "Query execution" for each of the constraint that they evaluate. Overall, a single feedback loop of the system, which mainly consists of the two methods initially last from one up to an average of 8 "Query executions" (~ 103-824 ms), then after a couple of feedback loops (with the utilization of the caches) the time decreases down the same time as a single "Query execution" (~ 103ms). Time measurements performed with Intel Core i5-8250U/8GB RAM using Chromium v96.0.4664.45.

4.3 Evaluation with Users

We conducted a preliminary and small scale task-based evaluation with users to see if users can use and/or like this interaction paradigm and for collecting feedback for improving the GUI, as well as the process. We used the 10 tasks shown in Table 1. Notice that the tasks are not trivial (like "find the x property of y"), but correspond to more complex information needs.

Task Selection. The first two tasks T1-2 require a simple query generation based from two examples. T3 requires the user to provide feedback to the system. T4-5 require the user to access more generated information than query results. T6 requires the use of the constrain examples instead of entity examples. T7-10 are more general and simulate tasks where the user have some preexisting knowledge and uses the system to retrieve related information.

Participants, Questionnaire and Results. We invited by email various persons to participate in the evaluation voluntarily. The users were asked to carry out the tasks and to fill (anonymously) the prepared questionnaire. No training material was given to

Table 1: Evaluation Tasks

ID	Task
T1	1. From the series of Batman movies, like "Batman Begins" and "The Dark Knight", try to find the names of other such movies
T2	2a. You know about "Before Sunset" and its sequel "Before Midnight", try to find the name of the third film of the series. (continues)
T3	2b. Try to find movies that have some properties in common with "Before" movies (you can utilize the NOTE 1 above).
T4	3a. What the movies "The Prestige" and "The Dark Knight" have in common other than actors. (continues)
T5	3b. Count how many movies the director and producer of "The Prestige" and "The Dark Knight" have made.
T6	4. Find the last Harry Potter movie "Harry Potter and the Deathly Hallows" and instead of selecting the movies, select only the actors "Daniel Radcliffe" and "Emma Watson". Count in how many movies both actors participate.
T7	5. In the movies "Agent Carter" and "Captain America: The First Avenger" we know that "Hayley Atwell" plays the character Agent Carter, try to find the name of other movies possibly containing the character. (note that the movie "The Duchess" has no relation to the character "Agent Carter")
T8	6a. Count how many movies are in the "Wolverine" series with the actor "Hugh Jackman". (continues)
T9	6b. What all the movies with "Wolverine" have in common.
T10	7. Find a movie with "Michael Caine" and "Leonardo DiCaprio"

them, just a paragraph with basic instructions and the participation to this evaluation was optional (invitation by email). Eventually, 22 persons participated (from April 14, 2022 to April 30, 2022). The number was sufficient for our purposes since, according to [2], 20 evaluators are enough for getting more than 95% of the usability problems of a user interface. In numbers, the participants were 22.7% female and 77.3% male, with ages ranging from 19 to 52 years; with 70% almost uniform distributed from 19 to 30. As regards occupation and skills, all have studied Computer Science (CS): 55% undergraduate and 45% postgraduate CS students.

The questionnaire is shown below, enriched with the results of the survey in the form of percentages written in bold:

- Q1 *How would you rate the "Keyword Search" tab?*: Very user friendly (**50%**), User friendly (**45.5%**), Not user friendly (**4.5%**), Very difficult to use (**0%**)
- Q2 *Rate the usability of the "Query Formulation" tab*: Very user friendly (**4.5%**), User friendly (**81.8%**), Not user friendly (**13.6%**), Very difficult to use (**0%**)
- Q3 *How would you describe the workflow?*: Very Intuitive (**9.1%**), Intuitive (**77.3%**), Unintuitive (**13.6%**), Very Unintuitive (**0%**)
- Q4 *How would you describe the constraint representation?*: Very Intuitive (**22.7%**), Intuitive (**59.1%**), Unintuitive (**18.2%**), Very Unintuitive (**0%**)
- Q5 *Would you prefer instead of the two tabs a single page with all the functionality?*: Yes (**40.9%**), Indifferent (**18.2%**), No (**40.9%**)
- Q6 *Would you use the app to formulate queries?*: Yes (**40.9%**), Maybe (**59.1%**), No (**0%**)
- Q7 *Would you use the app to find a movie?* Yes (**77.3%**), Maybe (**22.7%**), No (**0%**)
- Q8 *Have you ever formulated a SPARQL query?* Never (**22.7%**), Only a few times (without using SPARQL) (**13.6%**), Quite a lot (**63.6%**).
- Q9 *How would you rate the entire system?* Very Useful (**27.3%**), Useful (**72.7%**), Little Useful (**0%**), Not Useful (**0%**)
- Q10 *You can report here errors, problems, or recommendations.* (free text of unlimited length)

The results were very positive: By summing the two positive options (Very user-friendly/intuitive, user-friendly/intuitive), we can see that most users find it user-friendly (86.3% in Q2), find the workflow intuitive (86.4% in Q3), they liked the constraint representation (81.8% in Q3), they rated the system useful (100% in Q9), and it is interesting that many would use the system to find movies (Q7).

Task Performance. From the 220 collected responses (10 tasks x 22 participants), 65 (30%) reported failure to find the requested information. In most cases of the failed responses, the participants were able to find some answer which was either incomplete or wrong. Only in a few cases (6 responses, i.e. ~ 2%) the participants were unable to translate the task into actions for the system and were unable to find any answer. As shown in Figure 6, the participants faced problems mainly at task T8. The task did not imply a clear course of actions and so 8 participants (36%) ended up with correct answer, 10 (45%) with different answers and 4 (18%) failed to answer. From the 4 participants that had never used SPARQL, we have only 6 (15%) wrong responses however half of them reported that system is somewhat unintuitive to work with but still useful and they would maybe use it again in the future.

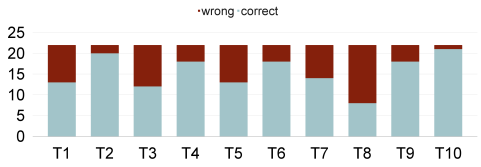

Figure 6: Success Rate by Task

5 CLOSING REMARKS.

We proposed a novel interactive method for SPARQL query formulation, for enabling users to formulate queries by providing *examples* and various kinds of *feedback*. The method can aid plain users and save effort from advanced users. The method can leverage other access methods (keyword search, browsing), it supports gradual formulation, and various kinds of interactive feedback. We described the algorithmic aspect and presented an interactive user interface that implements the approach. We have applied it in real datasets from DBpedia, and showcased the feasibility and the effectiveness of the approach. A running prototype is accessible through https://demos.isl.ics.forth.gr/SPARQL-QBE/. A task-based evaluation with users, that included users that are not familiar with SPARQL, provided evidence that the interaction is easy-to-grasp, intuitive and user-friendly and enabled the users to formulate the desired queries. The method could be applied not only to RDF, but on any graph database, and could be extended to various directions. There are several issues that are worth further research. For instance, at system level, one could easily extend the interactive system for specifying also the desired columns of the answer, as well as manual editing of the SPARQL query (e.g. for turning a URI of a triple pattern to a variable).

REFERENCES

[1] Marcelo Arenas, Gonzalo I Diaz, and Egor V Kostylev. 2016. Reverse engineering SPARQL queries. In *Proceedings of the 25th International Conference on World Wide Web*. 239–249.

[2] Laura Faulkner. 2003. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers* 35, 3 (2003), 379–383.

[3] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query from examples: An iterative, data-driven approach to query construction. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2158–2169.

[4] Steffen Metzger, Ralf Schenkel, and Marcin Sydow. 2013. Qbees: query by entity examples. In *Proceedings of the 22nd acm international conference on information & knowledge management*. 1829–1832.

[5] Moshé M Zloof. 1975. Query by example. In *Proceedings of the May 19-22, 1975, national computer conference and exposition*. 431–438.