Preprint of:
A. Akritidis and Y. Tzitzikas, Querying Knowledge Graphs through Positive and Negative Examples and Feedback, Journal of Intelligent Information Systems (JIIS), 2024 (accepted for publication).

Querying Knowledge Graphs through Positive and Negative Examples and Feedback

Akritas Akritidis[1,2] and Yannis Tzitzikas[1,2*]

[1]Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH), N. Plastira 100, Voutes, Heraklion, Greece.
[2]Department of Computer Science, University of Crete, Heraklion, Crete, Greece.

*Corresponding author(s). E-mail(s): tzitzik@ics.forth.gr;
Contributing authors: akritas@ics.forth.gr;

Abstract

The formulation of structured queries over Knowledge Graphs is not an easy task. To alleviate this problem, we propose a novel interactive method for SPARQL query formulation, for enabling users (plain and advanced) to formulate gradually queries by providing examples and various kinds of positive and negative feedback, in a manner that does not pre-suppose knowledge of the query language or the contents of the Knowledge Graph. In comparison to other example-based query approaches, distinctive features of our approach is the support of negative examples, and the positive/negative feedback on the generated constraints. We detail the algorithmic aspect and we present an interactive user interface that implements the approach. The application of the model on real datasets from DBpedia (Movies, Actors) and other datasets (scientific papers), showcases the feasibility and the effectiveness of the approach. A task-based evaluation that included users that are not familiar with SPARQL, provided positive evidence that the interaction is easy-to-grasp and enabled most users to formulate the desired queries.

Keywords: Query by Example, Knowledge Graphs

1 Introduction

The formulation of structured queries has always been considered a challenging task. Especially in Knowledge Graphs where there are no specific constraints or limitations regarding the structure of their contents. The user must either have a prior understanding of the current structure of the graph, or the query language to acquire that understanding. An additional difficulty is the syntax of the query language itself, that makes the formulation of queries from plain users infeasible. Consequently plain users need training to formulate queries. For these reasons, several methods and tools have been developed over the years that try to assist the user to formulate queries either by providing helpful information while the user writes the query or by hiding the text based nature of the query behind a dynamic visual interface. Most of these approaches benefit both plain and knowledgeable users in formulating SPARQL queries without requiring much familiarity with the knowledge graph or the query language or both.

In this paper we present "SPARQL by Example" an interactive example based method for formulating queries that utilizes a feedback loop. The approach is inspired by the *Query-by-Example* paradigm [33] that was developed in the context of Relational Databases, as well as by the *relevance feedback* mechanisms in Information Retrieval. The main idea is the following: The user provides one or more entities, that she has discovered while browsing or by keyword search. Then we compute a query whose result contains the provided entity, plus other ones that have commonalities with the entities provided by the user. Subsequently 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 more positive or negative examples.

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" using simple keyword search. The system then generates a list of selected 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, as shown in Figure 2. 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 given examples but ignore the actor.

Although there are some works that aim at offering a QBE-like interaction over Knowledge Graphs in RDF, i.e. [2, 17, 19], our approach has some distinctive characteristics. In particular, "Qbees" [19] neither supports negative

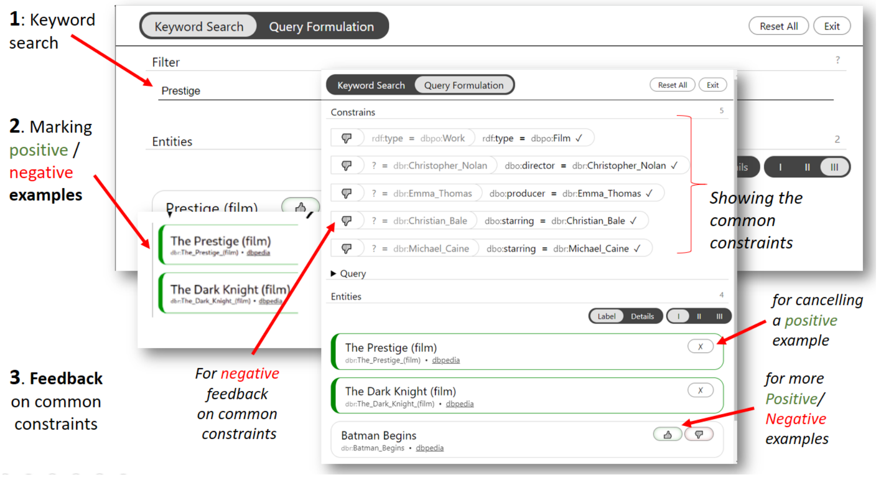

1: Keyword search
2. Marking positive / negative examples
3. Feedback on common constraints
For negative feedback on common constraints
Showing the common constraints
for cancelling a positive example
for more Positive/ Negative examples
Keyword Search
Query Formulation
Reset All
Exit
Filter
Prestige
Entities
Prestige (film)
The Prestige (film)
The Dark Knight (film)
Constrains
rdf:type = dbpo:Work
rdf:type = dbpo:Film ✓
? = dbr:Christopher_Nolan
dbo:director = dbr:Christopher_Nolan ✓
? = dbr:Emma_Thomas
dbo:producer = dbr:Emma_Thomas ✓
? = dbr:Christian_Bale
dbo:starring = dbr:Christian_Bale ✓
? = dbr:Michael_Caine
dbo:starring = dbr:Michael_Caine ✓
Query
Label
Details
Batman Begins
dbr:Batman_Begins • dbpedia

Fig. 1 Running Example: Finding movies through examples and feedback (via SPARQL-QBE)

```
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 .
}
```

Fig. 2 The formulated SPARQL example of the running example

examples nor produces a pure SPARQL query, "Query from examples" [17] requires the user to answer a number of questions, while "Reverse engineering SPARQL queries" [2] cannot receive feedback on the generated constraints.

In a nutshell, the contributions of this work are: (a) a novel interactive method for formulating SPARQL by example, (b) an analysis of the process, (c) an implementation of the model that proves its feasibility, and (d) an evaluation of the model with users with promising results. The basic idea has been described in the demo paper [1]. In comparison to that work, the current paper: (i) describes in more detail the related work and the placement of the current work, (ii) provides examples for making clear the benefits of the interaction, (iii) details the process, and the corresponding algorithmic aspects, (iv) includes application results from several datasets.

The rest of this paper is organized as follows: Section 2 describes the context and related work, Section 3 introduces the method and the corresponding algorithms. Section 4 describes the implementation of the approach, i.e. the system SPARQL-QBE, and discuss its expressive power and scalability. Section 5 is about evaluation, specifically it reports the results of a task-based evaluation with

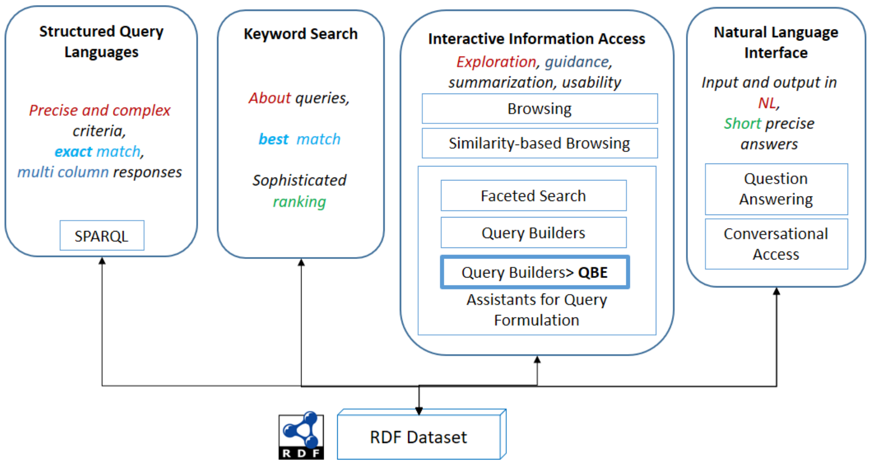

Structured Query Languages
Precise and complex criteria, exact match, multi column responses
SPARQL
Keyword Search
About queries, best match
Sophisticated ranking
Interactive Information Access
Exploration, guidance, summarization, usability
Browsing
Similarity-based Browsing
Faceted Search
Query Builders
Query Builders> QBE
Assistants for Query Formulation
Natural Language Interface
Input and output in NL, Short precise answers
Question Answering
Conversational Access
RDF Dataset
RDF

Fig. 3 An Overview of Access Methods over RDF

users, compares the functionality of the approach with other "example-based" systems, and discusses when the proposed method in beneficial in comparison to other methods for query formulation, as well as applicability and limitations. Finally Section 6 concludes the paper and identifies issues for further research.

2 Context and Related Work

Section 2.1 discusses the placement of this work in the spectrum of access methods over RDF, Section 2.2 discusses SPARQL formulators, Section 2.3 describes example-based SPARQL works, while Section 2.4 discusses cases from different domains where the notion of "example" is used.

2.1 Context. Overview of Access Methods over RDF

There are several access methods over RDF, as it can be seen in Fig. 3. In particular, they can be divided in four different categories, a) Structured Query Languages (like SPARQL and various extensions of the language), b) Keyword Search (e.g. [21]), c) Interactive Information Access that includes plain browsing, similarity-based browsing [4], faceted search [29], and query builders, and d) Natural Language Interfaces (like QA [20]).
Our Placement. Our work falls in the category of Interactive Information Access, specifically on assistants for query formulation.

2.2 SPARQL Query Formulators in General

The tools that aid the formulation of SPARQL queries could be divided into two categories *Query Builders* and *Faceted Search*.

Query Builders like SPARQL Assist [18] and YASGUI [24] simply assist the user to write the query with the main interface being a text field for SPARQL. More visual Query builders, like SPARKLIS [10], ResearchSpace

[22], Sparnatural [11], and Assistive Query Building (A-QuB) [16], enable the user to construct a query step by step using interactive graphical interfaces. For instance SPARKLIS [10], enables the user to inspect the schema and select elements through which to formulate the desired query. To hide the complexity of the schema, systems like ResearchSpace [22], Sparnatural [11], and Assistive Query Building (A-QuB) [16] enable the user to formulate a query over an abstraction of the schema, and assist the user by providing autocomplete fields, dropdown lists and other similar elements. The queries that are formulated over that view are then translated to SPARQL queries over the original schema. This approach is very useful in cases the ontology is big and complex (like CIDOC CRM [7]), since it would be almost impossible for the user to formulate directly one of the derived SPARQL queries. However, the abstraction of the model has to be configured by the designer. The formulators, like RDF Explorer [30] allow the user to freely explore an interactive visual graph, with the assistance of various components (search panel, visual query editor, node detail view, node editor, SPARQL query editor), that will lead to the formulation of a query (see [13] for an overview of older visual query builders published prior to 2015).

Faceted Search systems over RDF (see [29] for a survey), have a more exploratory nature, showing also the entities at each state and offering to the user only options that lead to non empty results. For a recent Faceted Search system over RDF Knowledge Graphs that supports also analytic queries, see [23]. Somehow relevant are the extensions of SPARQL with *similarity*, like [9, 31] enabling the formulation of criteria that would be hard to express in SPARQL.

Our Placement. Our work differs since the main input from the user a set of (positive and negative) entities.

2.3 Example-based SPARQL Query Formulators

There have been other works where examples are used as the starting point for finding a desired set of results. Although the term "example" is used in every case (i.e. in the context of each such work) the definition of the term is different. In our work, the notion of "example" has 4 different types: positive and negative entities, and positive and negative property-value pairs (constraints).

In "Query from examples" [17] the user provides examples in a form of dataset-result pairs, however they may not be part of the final results. The system then starts a feedback loop with the user by asking specific generated questions.

In "Qbees: query by entity examples" [19] the examples are entities but only positive, there is no support for negative examples. The results are directly generated based on ranking of common properties without the formulation of corresponding SPARQL query.

In "Reverse engineering SPARQL queries" [2] (and in works that use the same technique, i.e. SPARQLByE [6]), the examples are positive and negatives entities, there is no support for specific property-value pairs. The system

generates a query based on the given examples with an optional feedback loop where the user can adjust the given examples in every step.
Our Placement. Our work most resembles the "Reverse engineering SPARQL queries", however the techniques described in that paper cannot integrate property-value pairs as possible positive or negative examples. The addition of property-value pair examples, especially the negative ones, can greatly improve the feedback since it saves the user from having to find counter-examples. Counter-examples are difficult to find and may not be interpreted by the system as intended.

2.4 "Example"-based Approaches in other Domains and Tasks

Query by Example in Relational Databases. In the context of relational databases, almost 50 years ago, Moshé M. Zloof developed the Query-by-Example language [33]. The goal of the language was to allow non-programmers to query relational databases. The user is provided a visual area with empty tables where the user could add rows and fill cells. Every table provided corresponded to a table in the database with the same columns and column names but with none of the existing rows. In the simplest cases, the user constructs example rows with a combination of constants and variables in some of the columns. Then the system provides values for the variables that conform with the example rows. The variables can be referenced and are shared between all the example rows of each table which allows the creation of complex queries that support implicit joining of tables. The language was also extended to support more operations than just querying [34] [32]. A study of real users (of that era) trying to solve problems using Query-by-Example [27] has shown promising results. However the number of tables/columns of the database was relatively small in comparison to the number of concepts and relationships of a knowledge graph. For instance, if we map each concept to a table, and each incoming/outgoing property of that entity to a column of that table, then we would get a big number of tables, each having a big number of columns, something that would not be handy for direct manipulation through interaction by the user. Many graphical front-ends for databases use the ideas from QBE (Query By Example) today.

It should be noted that in Moshé M. Zloof's papers [33] the "examples" are effectively a combination of constraints that describe the expected results. Note that relational databases are value-based, however RDF is graph-based and has explicit entities. To this end, we use entities instead of values, and as we shall see this is more flexible (and economical) for the user than values, since an entity is associated with several values.

Information Retrieval (IR). In IR we have *Relevance Feedback* [26] where the user provides positive and negative feedback to the hits returned to the initial query, which are used for improving the query. The updated query is aimed at returning better results, however it is larger (and more expensive to evaluate), and can lead to some unexpected results (low explainability). In our case we do not have this problem, since we use the input for the common constraints and the user selectes the desired.

Schema Mappings. The notion of example has also been used for constructing mappings between schemas. For instance, according to the *ostensive method* for schema mappings [28], two systems exchange sets of objects and try to compute and return the best approximate queries that describe these objects, and this enables the two parties to establish mappings between their terms and queries.

3 SPARQL by Example: Problem Statement, Principles and Algorithms

Section 3.1 states the problem, Section 3.2 introduces the proposed process, that we call `SPARQL-QBE`, accompanied with examples, Section 3.3 describes extended constraints, and Section 3.4 provides the full algorithm.

3.1 Problem Statement

We shall first describe the general idea and principles, and then (in Section 3.2) we will provide examples.

Given a set of positive and negative examples, E_P and E_N respectively (where $E_P \cap E_N = \emptyset$), the objective is to compute a query q, such that answer of the q (strictly) contains the positive examples, i.e. $ans(q) \supset E_P$ and does not contain the negative examples, i.e. $E_N \cap ans(q) = \emptyset$. We confine ourselves to conjunctive queries, where each conjunct c is a property-value pair.

Of course there may be several such queries, or no query at all. Since the user provides the set E_P as a set of indicative examples, to provide him/her with *more* entities, it is reasonable to try finding those conditions that the elements of E_P satisfy, and suggest to the user those (more) entities that satisfy these constraints. Since there may be several common constraints, there is a need for methods for selecting those to be included in the query. Since we are in the context of an interactive system, this selection can be based on the following desiderata: (a) the selection should support *gradual enlargement* of the answer, (b) the selection should be *transparent* and *explainable* (i.e. the user should understand why he gets more entities). For this reason, we propose computing and showing the common constraints in a way that enables the user to declare any of the constraints as *wanted* or *unwanted* creating in this way the sets of positive and negative constraint examples, C_P and C_N respectively (where $C_P \cap C_N = \emptyset$).

As regards the negative entity examples, i.e. the entities in E_N, they are used to exclude certain combinations of common constraints. Note that it is possible for the entities of E_N to satisfy some of the common constraints but not all the constraints that will be included in the final query q. Essentially, we would like a function that takes as input the aforementioned examples, and returns the desired query q, i.e. `SPARQL-QBE`$(E_P, E_N, C_P, C_N) = q$

3.2 The Exact Process of `SPARQL-QBE`

Positive Examples. Let K be the set of RDF triples of the underlying Knowledge Graph. Let hereafter use E_P to denote the initial *positive* examples, i.e. one or more URIs selected by the user as indicative examples. Let $\mathcal{CC}(E_P)$ be the set of *common constraints* of the selected entities, defined as: $\mathcal{CC}(E_P) = \{ \ (p,v) \ \ | \ \ \forall e \in E_P, (e,p,v) \in K\}$ For example, assuming two movies $E_P = \{m1, m2\}$, we may have $\mathcal{CC}(E_P) = \{(director, Maria), (starring, James), (year, 2022)\}$. This is the initial query q_0, and we can show to the user the constraints of this query $q_0 = \mathcal{CC}(E_P) = \{c_1, \ldots, c_k\}$ as well as the answer of q_0, denoted by $ans(q_0)$, as shown in the upper right part of Figure 1.

Positive/Negative Feedback on Constraints. The user can proceed and provide *positive* or *negative feedback on the constraints*, i.e. the user can define the set of *wanted* and/or *unwanted* constraints, either from $\mathcal{CC}(E_P)$, and/or from inspecting the *descriptions* of the entities of the current answer, $ans(q_0)$. The latter set is defined as: $\mathcal{D}(ans(q_0)) = \{ \ (p,v) \ \ | \ \ (s,p,v) \in K \ \forall s \in ans(q_0) \ \}$ In this way the user can interactively select a set of *positive* constraints C_P and and a set of *negative* constraints C_N. Each is a subset of $\mathcal{CC}(E_P) \cup \mathcal{D}(ans(q_0))$, that is, the common constraints are not related only to E_P but also on the answer obtained by the first query. In our running example suppose that:
$C_P = \{(director, Maria), (genre, comedy)\}$ and $C_N = \{(starring, James)\}$. The first constraint in C_P comes from $\mathcal{CC}(E_P)$, while the second (i.e. (genre,comedy)), comes by inspecting $\mathcal{D}(ans(q_0))$. With this feedback we can define CC' as $CC' = (\mathcal{CC}(E_P) \cup C_P) \setminus C_N$, i.e. in our example:
$CC' = \{(director, Maria), (genrne, commedy), (year, 2022)\}$, and this is the new query, i.e. $q_1 = CC'$.

Negative Examples. So far we have seen negative feedback on constraints. Let's suppose the user has provided negative examples, i.e. a set E_N. We will use this to exclude from the powerset of CC' (i.e. from all possible conjunctive queries) those queries that would include any entity of E_N in their results. The exclusion will indirectly remove the negative examples from the results of the final query and possibly remove other entities too.

Continuing our example, suppose the user provides one negative example $E_N = \{m3\}$ where $\mathcal{D}(\{m3\}) = \{(director, George), (genre, comedy), (year, 2022)\}$. From the 8 possible subsets of $CC' = \{(director, Maria), (genre, comedy), (year, 2022)\}$ we exclude all possible subsets of $\mathcal{D}(\{m3\})$ found in CC' which are $\{(genre, comedy)\}$, $\{(year, 2022)\}$, their union $\{(genre, comedy), (year, 2022)\}$ and the empty set, so the remaining 4 subsets, let's call them *candidate queries*, are:

$$\begin{aligned} Q_1 = \{\{ \; & (director, Maria), (genre, comedy), (year, 2022)\}, \\ \{ \; & (director, Maria), (genre, comedy) \; \}, \\ \{ \; & (director, Maria), (year, 2022) \; \}, \\ \{ \; & (director, Maria) \; \}\} \end{aligned}$$

Now suppose the user gives one more negative example $m4$, therefore now we have $E_N = \{m3, m4\}$ where
$\mathcal{D}(\{m4\}) = \{(director, Maria), (genre, drama), (year, 2022)\}$. From this we have to exclude $\{(director, Maria)\}$, $\{(year, 2022)\}$ and their union, ending up to a Q_2 that has two possible queries:

$$\begin{aligned} Q_2 = \{\{ \; & (director, Maria), (genre, comedy), (year, 2022) \; \}, \\ \{ \; & (director, Maria), (genre, comedy) \; \}\} \end{aligned}$$

For these two possible queries, we need a ranking method to select one of them. So the full process (from a logical point of view) is (we use $P(C)$ to denote the powerset of C):

$$\begin{aligned} CC &= \mathcal{CC}(E_P) \\ C &= (CC \cup C_P) \setminus C_N \\ Q &= \{q \;\mid q \in P(C), \forall n \in E_N, n \notin ans(q)\} = \{q \;\mid q \in P(C), ans(q) \cap E_N = \emptyset\} \\ q &= Rank(Q) \end{aligned}$$

Ranking. As regards the last step, i.e. *ranking*, one approach is to select the query *with the higher number of constraints*, thus we could write: $q = \arg_q \max_{q \in Q}(\mid q \mid)$. Instead of getting the query with the max number of constraints, and also for resolving ties, we can rank the elements of Q according to the *expected size* of their answer. Below we describe two main methods for ranking the elements of Q: a fast but approximate method $\mathcal{M}_{freq}$ and a more accurate but more expensive method $\mathcal{M}_{conj}$.

Method $\mathcal{M}_{freq}$. At first we introduce some notations. We shall use pr^i to denote the number of instances of a property pr and analogously we use $(pr, vl)^i$ to denote the number of instances of a property pr with a value vl (where superscript i stands for instances):

$$\begin{aligned} pr^i &= \mid \{(s,p,o) \in K \;\mid\; p = pr\} \mid \\ (pr, vl)^i &= \mid \{(s,p,o) \in K \;\mid\; p = pr, o = vl\} \mid \end{aligned}$$

We can define the *frequency* of a constraint $c = (pr, vl)$ as $freq((pr, vl)) = \frac{(pr,vl)^i}{pr^i}$ We can use this metric to *rank* each constraint by either promoting the most frequent ones or the most rare ones (or the user could select the desired option at interaction time). In the running example of the introductory

Table 1 Frequencies in our running example

$freq((pr, vl))^{-1}$	pr	vl	$(pr, vl)^i$	pr^i
6884	'dbo:director'	'dbr:Christopher_Nolan'	15	103261
6423	'dbo:producer'	'dbr:Emma_Thomas'	10	64237
2554	'dbp:starring'	'dbr:Christian_Bale'	49	125130
1026	'dbp:starring'	'dbr:Michael_Caine'	122	125130

Table 2 The method $\mathcal{M}_{conj}$

$\mid ans(q) \mid$	q
4	'dbp:starring = Michael Caine, dbp:starring = Christian Bale, dbo:producer = dbr:Emma_Thomas, dbo:director = dbr:Christopher_Nolan, rdf:type = dbo:Film'
7	'dbp:starring = Michael Caine, dbo:producer = dbr:Emma_Thomas, dbo:director = dbr:Christopher_Nolan, rdf:type = dbo:Film'
9	'dbo:producer = dbr:Emma_Thomas, dbo:director = dbr:Christopher_Nolan, rdf:type = dbo:Film'
10	'dbo:producer = dbr:Emma_Thomas, rdf:type = dbo:Film'
15	'dbo:director = dbr:Christopher_Nolan, rdf:type = dbo:Film'
49	'dbp:starring = Christian Bale, rdf:type = dbo:Film'
122	'dbp:starring = Michael Caine, rdf:type = dbo:Film'

section, the frequencies are those shown in Table 1. For example 'Michael Caine' has starred in over double the number of movies than 'Christian Bale', which makes the corresponding constraints to have a different effect in the expected answer size, so if we want to gradually increase the answer size then we can select "Chistian Bale" (that appears in 49 movies), and then "Michael Caine" (that appears in 122 movies). It follows that we can use the ranking of $\mathcal{M}_{freq}$ in two ways. Firstly, for a $q \in Q$, we can define $score_{freq}(q) = \sum_{c \in q} freq(c)$, and we can rank in this way the elements of Q. Secondly, we can reduce the set Q by removing the queries $q \in Q$ which contain one of the lowest ranked constraints. The advantage of this method is that it is fast, the required frequencies can be pre-computed (like the histograms in relational databases [14]). One weakness of the method is that from these frequencies, we cannot predict accurately the impact of each constraint when combined with the others, i.e. the size of the answer of the formulated query.

Method $\mathcal{M}_{conj}$. To tackle the weakness of $\mathcal{M}_{freq}$, for each member q of Q, we can compute $\mid ans(q) \mid$ that is, the size of the answer if q is used. The advantage of this method, let's call it $\mathcal{M}_{conj}$, is that the user can see the size of the answer of that possible query, hence we could rank these suggested conjunctions of constraints, based on $\mid ans(q) \mid$, to offer a kind of "guaranteed query relaxation functionality", enabling the user to gradually get more entities. Table 2 shows the elements of Q ordered with respect to their answer size. We can see suggestions whose answer ranges from 4 to 122.

One weak point of this method is that it is more expensive than $\mathcal{M}_{freq}$ to compute, since it is exponential with respect to $\mid C \mid$, so it can be used only if $\mid C \mid$ is small. However, since the constraints of each query are combined by logical conjunction, the computation of $ans(q)$ is not expensive: we compute and keep stored $ans(\{c\})$ for each $c \in C$ where $\{c\}$ a query with a single constrain c, and compute $ans(q)$ by taking the intersection of all $ans(\{c\})$ for each $c \in q$ As regards ties, if two possible queries, say q and q' have the same answer size, we could select the one with the more constraints or the one with less constraints, to promote accordingly a more or less descriptive query based on the existing dataset, or show both options to allow the user to make that choice. Currently, we select the query with the most constraints which implicitly gives the user more information and control.

Proposed Ranking Policy: Let us now compare the time complexity of these two ranking methods. Let C be the set of common constraints computed at line 2 of Algorithm 1. With no special indexing or caching, to compute the ranking based on $\mathcal{M}_{freq}$, we have to compute $\mid C \mid$ simple (property-object) queries for getting the frequencies that are required, for approximating the answer size of each $q \in Q$. Consequently, the cost is $\mid C \mid$ simple (property-object) queries and $\mid Q \mid$ additions. Instead, for computing the ranking according to $\mathcal{M}_{conj}$, we have to compute $ans(q)$ for each query in Q. The size (in terms of property-object constraints) of each such query ranges from 1 to $\mid C \mid$. Consequently, in the worst case (where $\mid Q \mid = 2^{|C|}$) we have to evaluate $2^{|C|}$ queries. Therefore, at system level, it is better to start from $\mathcal{M}_{freq}$ (that is applicable even in big KGs) in order to remove some of the lowest ranked constraints. After reaching a number of constraints where Q is not that big, the system could switch to $\mathcal{M}_{conj}$.

Case: No Extra Entities. If $ans(q) = E_P$, i.e. if the user does not get any extra entities (apart from E_P), the system should try to *relax* the query. This can be achieved by deleting some of the constraints or even selecting a new set of constrains. To do so, we extend the use of $\mathcal{M}_{freq}$ and $\mathcal{M}_{conj}$. In case of $\mathcal{M}_{freq}$ we delete the constraint with the smallest frequency, and if the results of the query do not change, we repeat the process and delete more constrains until we see a change in the results. In case of $\mathcal{M}_{conj}$, we simply select the next query in the rank.

Case: Single Positive Example. If $Ep = \{e\}$, i.e. if the user has selected a single positive example, the system will consider all properties of e as the set of common constrains $CC(\{e\}) = \{\ (p,v)\ \mid\ (e,p,v) \in K\ \}$. Not all of them will be included in the generated query, those that will be included in the query depend on the two ranking methods described earlier. However the user, through the negative feedback, can direct the system to different constraints (if the top ranked ones are not the desired ones).

3.3 Extended Constraints

We have defined a constraint as property value pair and an entity e satisfies a constraint $c = (p, v)$ when (e, p, v) exists in the knowledge graph.

Operators. In order to extend the expressiveness of the constraints we include an operator for every constraint. The operator describes the condition between the value of the constraint and the value of the corresponding property of an entity that is required for the entity to satisfy the constraint. The previous

behaviour is denoted as the operator "=" so that (p, v) is now represented as $(p, =, v)$. We define the operators $\{<, \leq, \neq, \geq, >\}$ as the operations of the corresponding math operators.
Property paths. The second extension is the replacement of the first element of the constraint from a single property to a *property path.* The property path is defined similarly to a SPARQL property path where a constraint $c = (\{p1, p2\}, =, v)$ is satisfied by an entity e when a linking entity l exists such that both $(e, p1, l)$ and $(l, p2, v)$ also exist in the knowledge graph. We also extend the possible user feedback on the constraints to cover also property path-value pairs. Note that we cannot proactively compute such path-value pairs and display them to the user, since their number can be extremely high. For this reason, we let the user to expand a path towards the desired direction. A related screenshot is shown later in Figure 7.

3.4 Algorithm

We support a *session-based* interaction scheme, allowing the user to refine/change his/her query *gradually.* As stated earlier, the process starts by selecting (by browsing, keyword search or any other access method) at least two positive examples E_P. Then it enters into an interaction loop. The system accepts 4 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. Given a dataset and the 4 inputs, the system will output the same query q (stateless), 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, the user after inspecting the answer A of the current query, can either: (i) provide positive or negative feedback to the entities of the answer received by selecting elements of A, $E'_P = (E_P \cup E_{P_{added}}) \setminus E_{P_{deleted}}$, $E'_N = (E_N \cup E_{N_{added}}) \setminus E_{N_{deleted}}$), (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; they can also add such a constraint in negated form $(p \neq v)$, and get $C_{P'} = C_P \cup \{(p, op, v)\}$.

Algorithm 1 QBE Interactive Loop

```
1: function QBE-Interactive($E_P, E_N$: sets of entities, $C_P, C_N$: sets of constraints)
2:     $C \leftarrow (\mathcal{CC}(E_P) \cup C_P) \setminus C_N$
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 $A = ans(q)$
9:     Receive any input from the user and get $E'_P, E'_N, C'_P, C'_N$
10:    call QBE-Interactive ($E'_P, E'_N, C'_P, C'_N$)
11: end function
```

4 Application

At first we describe the implementation of the model (Section 4.1), and then we discuss its expressive power (Section 4.2), and scalability (Section 4.3).

4.1 Functionality and Implementation of SPARQL-QBE

We have implemented a proof-of-concept prototype that we call SPARQL-QBE. The application supports the scenario described in section 1. It starts in a keyword search mode for enabling the user to find the starting examples, an example is shown in Figure 4.

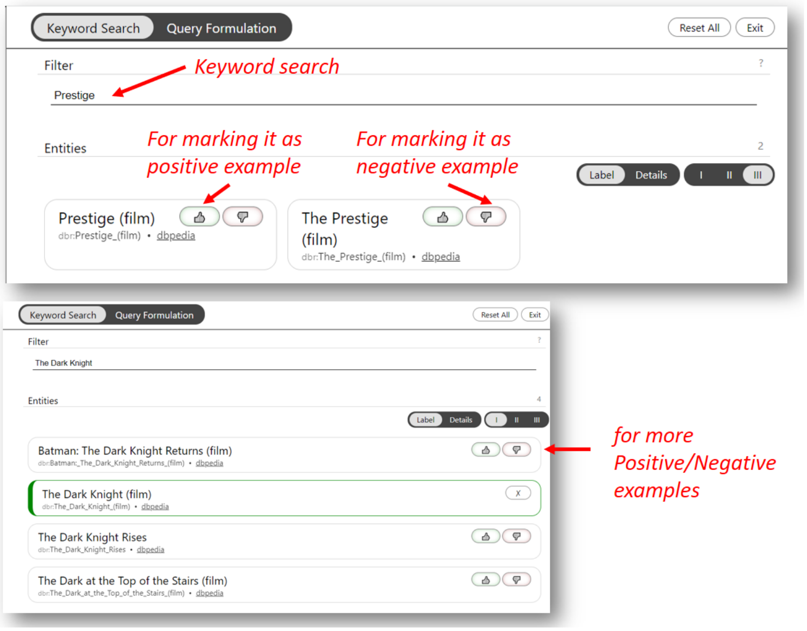
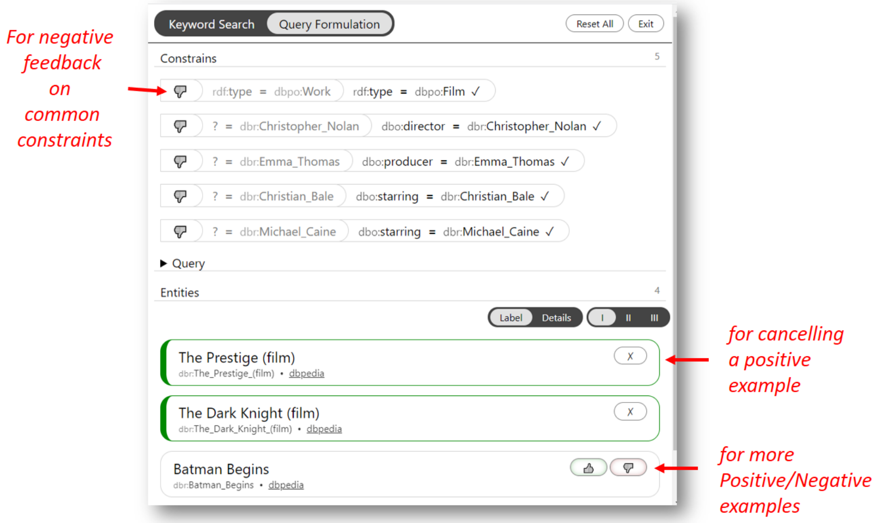

Keyword Search
Query Formulation
Reset All
Exit
Filter
Keyword search
Prestige
Entities
For marking it as positive example
For marking it as negative example
2
Label
Details
I
II
III
Prestige (film)
dbr:Prestige_(film) • dbpedia
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia
Keyword Search
Query Formulation
Reset All
Exit
Filter
The Dark Knight
Entities
4
Label
Details
I
II
III
Batman: The Dark Knight Returns (film)
dbr:Batman:_The_Dark_Knight_Returns_(film) • dbpedia
The Dark Knight (film)
dbr:The_Dark_Knight_(film) • dbpedia
X
The Dark Knight Rises
dbr:The_Dark_Knight_Rises • dbpedia
The Dark at the Top of the Stairs (film)
dbr:The_Dark_at_the_Top_of_the_Stairs_(film) • dbpedia
for more Positive/Negative examples

Fig. 4 SPARQL-QBE: Step 1: Keyword Search and marking positive and negative examples

The user can continue in this way for providing more examples, as shown in Figure 4(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 5.

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 6.

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 7. The user can then create a property path constraint by selecting one


Keyword Search
Query Formulation
Reset All
Exit
For negative feedback on common constraints
for cancelling a positive example
for more Positive/Negative examples
Query
Label
Details
I
II
III

Constrains 5

rdf:type = dbpo:Work | rdf:type = dbpo:Film ✓
? = dbr:Christopher_Nolan | dbo:director = dbr:Christopher_Nolan ✓
? = dbr:Emma_Thomas | dbo:producer = dbr:Emma_Thomas ✓
? = dbr:Christian_Bale | dbo:starring = dbr:Christian_Bale ✓
? = dbr:Michael_Caine | dbo:starring = dbr:Michael_Caine ✓

Entities 4

The Prestige (film) — dbr:The_Prestige_(film) • dbpedia — X
The Dark Knight (film) — dbr:The_Dark_Knight_(film) • dbpedia — X
Batman Begins — dbr:Batman_Begins • dbpedia

Fig. 5 SPARQL-QBE: Step 2: Negative feedback on common constraints, and more positive/negative examples

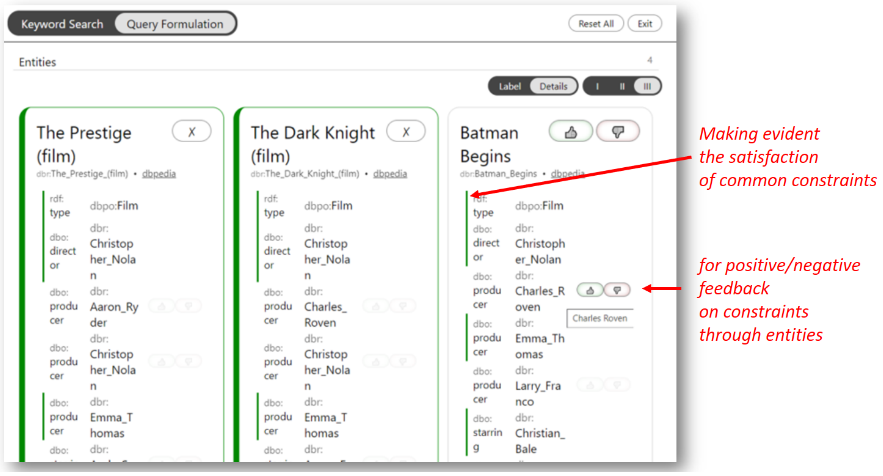
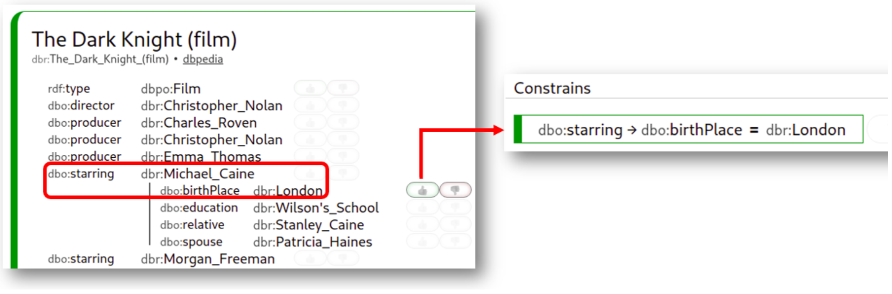

Keyword Search
Query Formulation
Reset All
Exit
Entities
4
Label
Details
I
II
III
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia
X
rdf:type dbpo:Film
dbo:director dbr:Christopher_Nolan
dbo:producer dbr:Aaron_Ryder
dbo:producer dbr:Christopher_Nolan
dbo:producer dbr:Emma_Thomas
The Dark Knight (film)
dbr:The_Dark_Knight_(film) • dbpedia
X
rdf:type dbpo:Film
dbo:director dbr:Christopher_Nolan
dbo:producer dbr:Charles_Roven
dbo:producer dbr:Christopher_Nolan
dbo:producer dbr:Emma_Thomas
Batman Begins
dbr:Batman_Begins • dbpedia
rdf:type dbpo:Film
dbo:director dbr:Christopher_Nolan
dbo:producer dbr:Charles_Roven
Charles Roven
dbo:producer dbr:Emma_Thomas
dbo:producer dbr:Larry_Franco
dbo:starring dbr:Christian_Bale
Making evident the satisfaction of common constraints
for positive/negative feedback on constraints through entities

Fig. 6 SPARQL-QBE: Making evident constraint satisfaction, and constraint feedback through entities

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.

Implementation. 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 text files packaged with the application that the user loads once with the first load of the application.



The Dark Knight (film)
dbr:The_Dark_Knight_(film) • dbpedia

rdf:type	dbpo:Film
dbo:director	dbr:Christopher_Nolan
dbo:producer	dbr:Charles_Roven
dbo:producer	dbr:Christopher_Nolan
dbo:producer	dbr:Emma_Thomas
dbo:starring	dbr:Michael_Caine
dbo:birthPlace	dbr:London
dbo:education	dbr:Wilson's_School
dbo:relative	dbr:Stanley_Caine
dbo:spouse	dbr:Patricia_Haines
dbo:starring	dbr:Morgan_Freeman

Constrains

dbo:starring → dbo:birthPlace = dbr:London

Fig. 7 The extension of `SPARQL-QBE` with path expressions

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. These triples describe 112,668 films and 43,157 actors. More statistics about this dataset are given in the second column of Table 3. A deployment of `SPARQL-QBE` with this dataset is accessible through https://demos.isl.ics.forth.gr/SPARQL-QBE/(dataset option: MoviesActors). Note that several works include KGs for movie recommendations, e.g. [25].

Table 3 Statistics of the Dataset about Films and Actors, and the Dataset about Papers

Metric	Dataset about Films and Actors	Dataset about Papers
# triples	1,083,029	513,426
# entities	155,825	23,548
# unique types	2 (dbo:Film, dbo:Actor)	1 (dbo:Paper)
# unique subjects	155,825	23,548
# unique properties	118	4 (dbp:cited, rdf:type, rdfs:label, dbp:ref)
# unique values	322,270	207,448
# unique subject-property pairs	707,236	87,974
# unique property-value pairs	349,867	221,965

Efficiency. Overall, for the datasets described earlier, we have real time interaction. The following time measurements performed with Intel Core i5-8250U/8GB RAM using Chromium v96.0.4664.45.

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 $\sim$ 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 $\mathcal{M}_{conj}$ and $\mathcal{M}_{freq}$ 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" ($\sim$ 103-824 ms), then after a couple of feedback loops (with the utilization of the caches) the time decreases down to the same time as a single "Query execution" ($\sim$ 103ms). An analysis of the time (with and without caching) is shown in Figure 8, where we can see the impact of caching for the ranking methods.

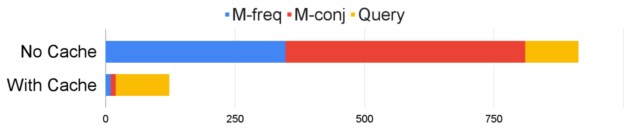


Fig. 8 Single feedback loop times in milliseconds (with and without caching)

4.2 Expressive Power

The system enables the formulation of single variable single class conjunctive queries. Although the system supports different classes, all the examples must be of the same class (or have a common superclass), and the first constraint of the generated query will refer to that class. Note that the system is designed with the single class query in mind since from a collection of positive examples that belong to different classes it would not be clear how to interpret such a feedback. Apart from the class restrictions, the generated queries consist of a set of constrains in conjunction where each constrain can be a positive or negative match of property with a single value. With the extended constraints, more general conditions with property paths and values are supported, meaning that even if all the examples are of the same class, the property paths indirectly allow for constraints that relate to multiple other classes.

The simplicity of the form, even with the extended constraints, allows for a straightforward visual representation of the query as a single list of constraints. However the conjunctions are between constraints that are not required to match to a single SPARQL statement. Some of that constrains could be a hidden complex SPARQL statement labeled with a user friendly notation without changing the feedback interface (eg. "$(year \geq 1980 \wedge year \leq 1989) \vee year = 80s$" labeled as "year = 80s").

4.3 Scalability

All operations have execution times that linearly depend on the number of entities and properties that are given as examples. Referring to the lines of the algorithm as shown in Alg. 1: Line 2 performs linear operations in the given input sets. Line 3 generates the set of valid queries. In worst case the size of the generated set is equal to the number of valid constraints raised to the power of 2. For the negative example check, there is no need for the query to be executed in the whole dataset since using the negative examples set as the target dataset is sufficient. Line 4 uses the two ranking methods as described in the corresponding sections $\mathcal{M}_{freq}$ and $\mathcal{M}_{conj}$, the methods require a query execution for every valid constraint and valid query accordingly, but they can be partially precomputed or cached during a session. Lines 5-7 perform a single query execution at each iteration of the loop. In the worst case there will be as many iterations as the number of valid queries. Lines 8-10 describe the interface and feedback operations and have no significant computation cost.

Ranking optimization: The first ranking method $\mathcal{M}_{freq}$ does not require whole queries and can also provide a ranking to single constraints. In order to limit the exponential nature of the power set of Line 3, we perform an initial reduction of the valid constraints using $\mathcal{M}_{freq}$.

In our JavaScript application we implement our own query execution operation which supports only the subset of SPARQL that can be generated. Since the formulated queries are single variable conjunctive queries (see previous section) this operation linearly depends on the number of entities.

5 Evaluation

Section 5.1 reports the results of a task-based evaluation with users. Section 5.2 compares the functionality of the approach with other "example-based" systems. Section 5.3 discusses when the proposed method in beneficial in comparison to other methods for query formulation, as well as applicability and limitations.

5.1 Task-based Evaluation with Users

We conducted a preliminary and small scale task-based evaluation with users for checking whether 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 4.

Table 4 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 (hinted that if all results are flagged as wanted, the query will extend to contain more results).
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"

Task Selection. Notice that the tasks are not trivial (like "find the x property of y"), but correspond to more complex information needs. The first two tasks T1-2 require a simple query generation based on two examples. T3 requires the user to give a feedback, i.e. to perform an action during the feedback loop. T4-5 require the user to access more generated information than query results. T6 requires the use of the constraint examples instead of just entity examples. T7-10 are more general and simulate tasks where the user have some pre-existing 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 them, just a paragraph with basic instructions[1] 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 [8], 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 uniformly distributed from 19 to 30. As regards occupation and skills, all have studied Computer Science. In detail, 55% were undergraduate students, 41% of them postgraduate computer science students, and the rest computer engineers, professionals and researchers. Some participants were familiar with SPARQL however none were familiar with the given knowledge graph. 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%**).

[1] Like the help page of the prototype.

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)

5.1.1 Results Analysis and Discussion

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. As regards *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. $\sim$ 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 9, 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.

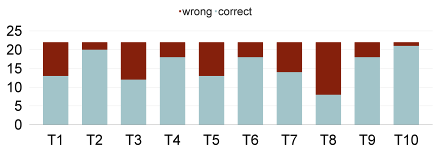


Fig. 9 Success Rate by Task

Free form Feedback. With respect to the *free form feedback* (Q10), the users provided some suggestions about some icons and omissions related to the dataset, not the system (e.g. omission of movies' genre and list of characters).

5.2 Comparison with systems that support the notion of "Example"

Compared to example-based systems like "Query from examples" [17], 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" [2], that work cannot

Table 5 Comparison with other Example-based Systems

System	Positive Entity Examples	Negative Entity Examples	Constraints Feedback
Qbees: query by entity examples [17]	✓		
Reverse engineering SPARQL queries [2]	✓	✓	
`SPARQL-QBE`	✓	✓	✓

receive feedback on the generated constraints, therefore has lower interactivity. Note that some works, i.e. [2] and [6], produce queries that contain the OPTIONAL operator of SPARQL. In `SPARQL-QBE` we do not produce such queries since the examples given by the user are entities or constraints where there are no null values, therefore the OPTIONAL operator is not needed.

5.3 Analytical Comparison

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. A comparison follows.

Comparing to Relational QBE. In comparison to QBE approached over relational databases, where the user has to fill in values in existing tables (after first selecting these tables), in our case (in graph databases in general) we do not have this limitation.

Comparing to non Example-Based Approaches (e.g. with Faceted Search). In a non example-based approach, like Faceted Search, 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. Indeed, there are information needs where it is easier for the user to point to examples than to find the conditions that entities should satisfy, e.g. a user may point to two songs of a similar style of music without knowing the actual name of that style. In addition, even if the user can point to the desired constraints, the number of them could be more than the number of examples that it is required through an example-based method.

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.

The Value of Negative Feedback. We should note here that without the possibility of providing negative feedback the system would behave like a hill climbing algorithm [15] and the user could be trapped to the first hill. From this perspective, the negative feedback can be conceived as a "step down" operation. In practice the user does not realize this "step down" since the interaction contains the sequence *Show Results* > *Negative Feedback* > *Step Down* (remove a constraint from the formulated query) > *Step Up* (new query considering the feedback) > *Show Results*. We should also mention that since the system makes decisions (as regards the common constraints to show) without asking the user, and this selection is based on frequencies, there should be a way to "undo" these decisions otherwise we will always end up with the constraints determined the criterion used by the system for constraint ranking.

Applicability. The method has wide applicability. Not only it can be used as a stand-alone method, but it can complement, browsing and keyword search systems, e.g. the user can first pick the desired entities (and put them in a cart) while browsing or searching with keywords, and then initiate the QBE process.

To what kind of Datasets is this Suitable? By testing this interaction over various datasets, we have realized that this approach is beneficial for the user, in comparison to other approaches, in cases where the user is not sure about the constraints that should be added to the desired query.

Apart from the movie-actors dataset, we generated a dataset with information about *published papers* where each paper is described by three properties: title, referenced papers and cited-by papers. In total, this dataset contains 513,426 triples, detailed statistics are given in the third column of Table 3.

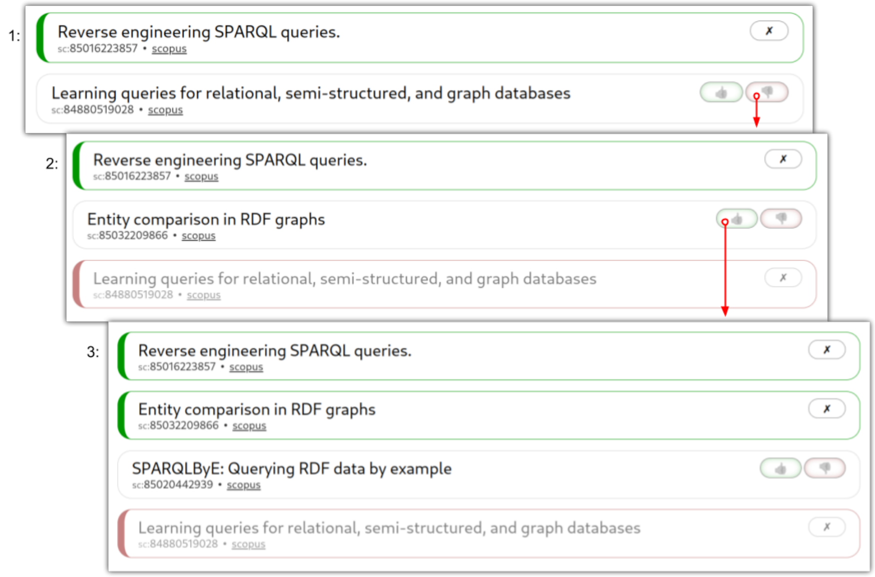
The user with `SPARQL-QBE` over this dataset can find papers based on some common properties of the example papers given.

In this context and through negative feedback, the system behaves more like a paper discovery system than a query generator. An indicative scenario, is described next, also shown in Figure 10:

1: The user starts with a reverse engineer SPARQL related paper, the system based on the papers' references generates a query that also returns a paper related to queries. Assuming that the user wants papers specific to SPARQL, flags the suggested paper as unwanted.

2: The system, based on the feedback, generates a new query that now suggests an RDF related paper. The paper may not be about SPARQL but it's related, so this time the user flags the paper as wanted.

3: The system finally generates a query that suggests a SPARQL related paper. The user could then continue this process until the system cannot suggest any new papers or restart the process with one of the papers that have been discovered.



1:

Reverse engineering SPARQL queries.
sc:85016223857 • scopus

Learning queries for relational, semi-structured, and graph databases
sc:84880519028 • scopus

2:

Reverse engineering SPARQL queries.
sc:85016223857 • scopus

Entity comparison in RDF graphs
sc:85032209866 • scopus

Learning queries for relational, semi-structured, and graph databases
sc:84880519028 • scopus

3:

Reverse engineering SPARQL queries.
sc:85016223857 • scopus

Entity comparison in RDF graphs
sc:85032209866 • scopus

SPARQLByE: Querying RDF data by example
sc:85020442939 • scopus

Learning queries for relational, semi-structured, and graph databases
sc:84880519028 • scopus

Fig. 10 Discovery scenario with the papers dataset

Limitations. One limitation of the proposed approach is that it cannot suggest any common constraint in case the positive examples have nothing in common. In such cases, constraints that have the form of path expressions could be identified and recommended to the user, or the user could expand one or more values of an entity for giving positive/negative feedback on values that are not directly connected with the entities of interest. This is an interesting direction for further research, one could investigate the applicability of techniques coming from works on RDF summarization [3, 12] and on Least Common Subsumers [5].

6 Concluding Remarks

We proposed a novel interactive method for SPARQL query formulation, for enabling users (plain and advanced) to formulate queries by providing examples and various kinds of positive and negative feedback, in a manner that does not pre-suppose knowledge of the query language or the contents of the Knowledge Graph. We have detailed the algorithmic aspects and presented an interactive user interface that implements the approach[2]. We have applied it in real datasets from DBpedia (Movies, Actors, etc), and other datasets (scientific papers), and showcased the feasibility (in terms of efficiency, i.e. real-time

[2]A running prototype is accessible through https://demos.isl.ics.forth.gr/SPARQL-QBE/

interaction), and the effectiveness of the approach. As regards the latter, a task-based evaluation that included users that are not familiar with SPARQL, provided evidence that that the interaction is easy-to-grasp and enabled the users to formulate the desired queries. In particular, all (100%) of the users rated the system useful, most users (86.3%) find it user-friendly, and only 2% of the participants were unable to translate an evaluation task into actions for the system. The method could be applied not only to RDF, but on any graph database and could complement and combined with the existing access methods (i.e. keyword search and browsing).

There are several issues that are worth further work and 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). Another interesting case (as described in the limitations), is to extend the system for enabling the user to give positive/negative feedback on path expressions.

Declarations

• Ethical Approval: Not applicable

• Availability of supporting data: The code of the running prototype, is publicly accessible (https://demos.isl.ics.forth.gr/SPARQL-QBE/). With "View Page Source" one can see all the source code.

• Competing interests: None

• Funding: FORTH-ICS

• Authors' contributions: All authors (Akritas Akrititis and Yannis Tzitzikas) contributed to the study conception, design and writing of this work, including the main manuscript text and the figures. Akritas Akrititis implemented the prototype system. All authors reviewed the manuscript.

• Acknowledgements: FORTH-ICS

References

[1] Akritidis, A., Tzitzikas, Y.: Demonstrating interactive SPARQL formulation through positive and negative examples and feedback. In: Demo Paper, 26th International Conference on Extending Database Technology, EDBT 2023 (2023). https://doi.org/10.48786/edbt.2023.71

[2] Arenas, M., Diaz, G.I., Kostylev, E.V.: Reverse engineering SPARQL queries. In: Proceedings of the 25th International Conference on World Wide Web (2016). https://doi.org/10.1145/2872427.2882989

[3] Čebirić, Š., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., Zneika, M.: Summarizing semantic graphs: a survey. The VLDB journal **28**, 295–327 (2019). https://doi.org/10.1007/s00778-018-0528-3

[4] Chatzakis, M., Mountantonakis, M., Tzitzikas, Y.: RDFsim: similarity-based browsing over DBpedia using embeddings. Information **12**(11) (2021). https://doi.org/10.3390/info12110440

[5] Colucci, S., Donini, F.M., Giannini, S., Di Sciascio, E.: Defining and computing least common subsumers in rdf. Journal of Web Semantics **39** (2016). https://doi.org/10.1016/j.websem.2016.02.001

[6] Diaz, G., Arenas, M., Benedikt, M.: SPARQLByE: Querying RDF data by example. Proceedings of the VLDB Endowment **9**(13) (2016). https://doi.org/10.14778/3007263.3007302

[7] Doerr, M.: The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. AI magazine **24**(3) (2003). https://doi.org/10.1609/aimag.v24i3.1720

[8] Faulkner, L.: Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. Behavior Research Methods, Instruments, & Computers **35**(3) (2003). https://doi.org/10.3758/BF03195514

[9] Ferrada, S., Bustos, B., Hogan, A.: Extending SPARQL with similarity joins. In: International Semantic Web Conference (2020). https://doi.org/10.1007/978-3-030-62419-4_12. Springer

[10] Ferré, S.: Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. Semantic Web **8**(3) (2017). https://doi.org/10.3233/SW-150208

[11] Francart, T.: Sparnatural. https://sparnatural.eu/ (2021)

[12] Goasdoué, F., Guzewicz, P., Manolescu, I.: Rdf graph summarization for first-sight structure discovery. The VLDB journal **29** (2020). https://doi.org/10.1007/s00778-020-00611-y

[13] Grafkin, P., Mironov, M., Fellmann, M., Lantow, B., Sandkuhl, K., Smirnov, A.V.: SPARQL query builders: Overview and comparison. In: BIR Workshops (2016)

[14] Ioannidis, Y.: The history of histograms (abridged). In: Proceedings 2003 VLDB Conference (2003). https://doi.org/10.1016/B978-012722442-8/50011-2. Elsevier

[15] Jacobson, S.H., Yücesan, E.: Analyzing the performance of generalized hill climbing algorithms. Journal of Heuristics **10** (2004). https://doi.org/10.1023/B:HEUR.0000034712.48917.a9

[16] Kritsotakis, V., Roussakis, Y., Patkos, T., Theodoridou, M.: Assistive query building for semantic data. In: SEMANTICS Posters&Demos (2018)

[17] Li, H., Chan, C.-Y., Maier, D.: Query from examples: An iterative, data-driven approach to query construction. Proceedings of the VLDB Endowment **8**(13) (2015). https://doi.org/10.14778/2831360.2831369

[18] McCarthy, L., Vandervalk, B., Wilkinson, M.: SPARQL assist language-neutral query composer. BMC bioinformatics **13**(1), 1–9 (2012). https://doi.org/10.1186/1471-2105-13-S1-S2

[19] Metzger, S., Schenkel, R., Sydow, M.: Qbees: query by entity examples. In: Proceedings of the 22nd ACM International Conference on

Information & Knowledge Management (2013). https://doi.org/10.1007/s10844-017-0443-x

[20] Nikas, C., Fafalios, P., Tzitzikas, Y.: Open domain question answering over knowledge graphs using keyword search, answer type prediction, SPARQL and pre-trained neural models. In: Proceedings of the 20th International Semantic Web Conference (2021). https://doi.org/10.1007/978-3-030-88361-4_14. Springer

[21] Nikas, C., Kadilierakis, G., Fafalios, P., Tzitzikas, Y.: Keyword search over RDF: Is a single perspective enough? Big Data and Cognitive Computing **4**(3), 22 (2020). https://doi.org/10.3390/bdcc4030022

[22] Oldman, D., Tanase, D.: Reshaping the knowledge graph by connecting researchers, data and practices in ResearchSpace. In: International Semantic Web Conference (2018). https://doi.org/10.1007/978-3-030-00668-6_20. Springer

[23] Papadaki, M.-E., Tzitzikas, Y.: RDF-ANALYTICS: Interactive analytics over rdf knowledge graphs. In: Demo Paper, 26th International Conference on Extending Database Technology, EDBT 2023 (2023). https://doi.org/10.48786/edbt.2023.70

[24] Rietveld, L., Hoekstra, R.: The YASGUI family of SPARQL clients. Semantic Web **8**(3) (2017). https://doi.org/10.3233/SW-150197

[25] Sacenti, J.A., Fileto, R., Willrich, R.: Knowledge graph summarization impacts on movie recommendations. Journal of Intelligent Information Systems **58**(1) (2022). https://doi.org/10.1007/s10844-021-00650-z

[26] Salton, G., Buckley, C.: Improving retrieval performance by relevance feedback. Journal of the American society for information science **41**(4) (1990). https://doi.org/10.1002/(SICI)1097-4571(199006)41:4⟨288::AID-ASI8⟩3.0.CO;2-H

[27] Thomas, J.C., Gould, J.D.: A psychological study of query by example. In: Proceedings of the May 19-22, 1975, National Computer Conference and Exposition. AFIPS '75. ACM, New York (1975). https://doi.org/10.1145/1499949.1500035

[28] Tzitzikas, Y., Meghini, C.: Ostensive automatic schema mapping for taxonomy-based peer-to-peer systems. In: International Workshop on Cooperative Information Agents (2003). https://doi.org/10.1007/978-3-540-45217-1_6. Springer

[29] Tzitzikas, Y., Manolis, N., Papadakos, P.: Faceted exploration of RDF/S datasets: a survey. Journal of Intelligent Information Systems **48**(2) (2017). https://doi.org/10.1007/s10844-016-0413-8

[30] Vargas, H., Aranda, C.B., Hogan, A.: RDF Explorer: A visual query builderja for semantic web knowledge graphs. In: ISWC Satellites (2019). https://doi.org/10.1007/978-3-030-30793-6_37

[31] Zheng, W., Zou, L., Peng, W., Yan, X., Song, S., Zhao, D.: Semantic SPARQL similarity search over RDF knowledge graphs. Proceedings of the VLDB Endowment **9**(11) (2016). https://doi.org/10.14778/2983200.2983201

[32] Zloof, M.M.: Query-by-example: A data base language. IBM Systems Journal **16**(4), 324–343 (1977). https://doi.org/10.1147/sj.164.0324

[33] Zloof, M.M.: Query by example. In: Proceedings of the May 19-22, 1975, National Computer Conference and Exposition (1975a)

[34] Zloof, M.M.: Query-by-Example: The invocation and definition of tables and forms. In: Proceedings of the 1st Intern. Conf. on Very Large Data Bases. VLDB '75. ACM, New York (1975b). https://doi.org/10.1145/1282480.1282482