Interactive Formulation of SPARQL Queries through Examples and Feedback

Akritas Akritidis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Yannis Tzitzikas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Interactive Formulation of SPARQL Queries through Examples and Feedback

Thesis submitted by
Akritas Akritidis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: [illegible]
Akritas Akritidis

Committee approvals: [illegible]
Yannis Tzitzikas
Professor, Thesis Supervisor

[illegible]
Dimitris Plexousakis
Professor, Committee Member

[illegible]
Kostas Magoutis
Associate Professor, Committee Member

Departmental approval: [illegible]
Polyvios Pratikakis
Assistant Professor, Director of Graduate Studies

Heraklion, March 2023

Interactive Formulation of SPARQL Queries through Examples and Feedback

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, and enable plain users to formulate SPARQL queries (and aid advanced users to formulate queries with less effort), in this thesis we introduce "SPARQL by Example", an interactive method for formulating SPARQL queries. 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 analyze this problem, we propose an interactive process, we present `SPARQL-QBE` a tool that implements this approach, and finally we report the results of a task-based evaluation with users that provided positive evidence about the usability of the approach.

Διαδραστική Διατύπωση Επερωτήσεων SPARQL μέσω Παραδειγμάτων και Ανατροφοδότησης

Περίληψη

Η διατύπωση δομημένων επερωτήσεων σε Γνωσιακούς Γράφους αποτελεί πρόκληση, καθώς προϋποθέτει γνώση και εξοικείωση με το συντακτικό της γλώσσας επερωτήσεων και τα περιεχόμενα του Γνωσιακού Γράφου. Για να απαλυνουμε αυτό το πρόβλημα και να επιτρέψουμε σε απλούς χρήστες να διατυπώνουν εύκολα επερωτήσεις SPARQL (και να βοηθήσουμε τους έμπειρους χρήστες ώστε να διατυπώνουν επερωτήσειςμε λιγότερη προσπάθεια), σε αυτήν τη διατριβή εισάγουμε τη μέθοδο «SPARQL by Example», μια διαδραστική μέθοδο για τη διατύπωση επερωτήσεων SPARQL. Βάσει αυτής, ο χρήστης υποδεικνύει θετικά ή/και αρνητικά παραδείγματα (οντότητες), το σύστημα υπολογίζει μια επερώτηση που περιγράφει αυτές τις οντότητες και στη συνέχεια ο χρήστης βελτιώνει την επερώτηση διαδραστικά παρέχοντας θετική/αρνητική ανατροφοδότηση για οντότητες και προτεινόμενες συνθήκες. Αναλύουμε αυτό το πρόβλημα, προτείνουμε μια διαδραστική διαδικασία, και παρουσιάζουμε το SPARQL-QBE ένα διαδικτυακό σύστημα που εφαρμόζει αυτήν την προσέγγιση. Τέλος αναφέρουμε τα αποτελέσματα μιας αξιολόγησης βάσει εργασιών με χρήστες που καταδεικνύουν τα θετικά στοιχεία (χρηστικότητα) αυτής της προσέγγισης.

Ευχαριστίες

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The formulation of structured queries has always been considered a challenging task. Especially in knowledge graphs where there are no specific constraints and limitation for the structure of their contents. The user must either have prior understanding of the current structure of the graph or either use the query language to acquire that understating. An additional layer of difficulty is also the syntax of the query language itself, which for plain users may require training. For these reasons, many methods and tools have been developed over the years that try to assist the user to formulate queries either by provide 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 thesis 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 [28] that was developed in the context of Relational Databases, as well as by the *relevance feedback* mechanisms in Information Retrieval [21]. The main idea is the following: The user provides one or more entities, that they 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.

To grasp the idea, Figure 1.1 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

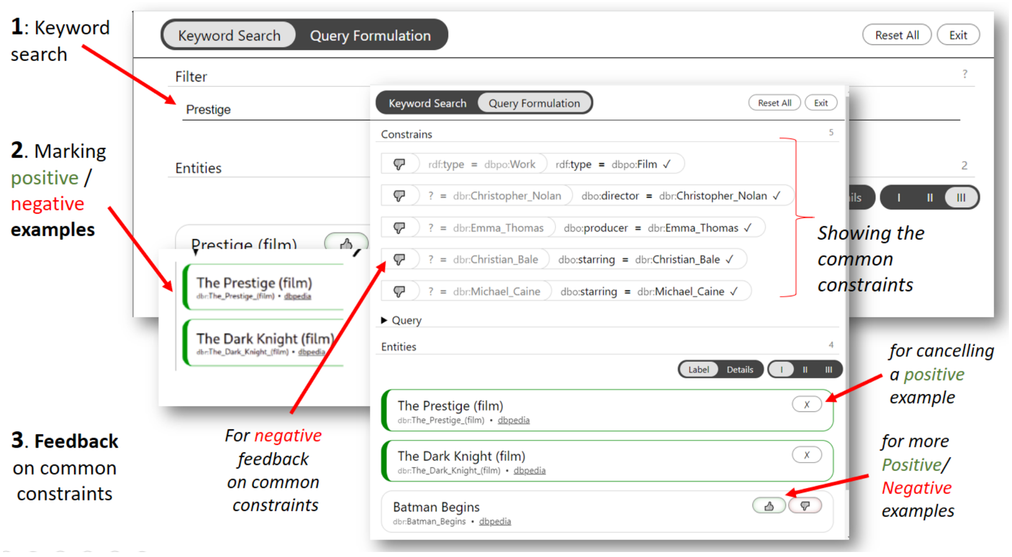

1: Keyword search
2. Marking positive / negative examples
3. Feedback on common constraints
For negative feedback on common constraints
Showing the common constraints
for cancelling a positive example
for more Positive/ Negative examples
Keyword Search
Query Formulation
Reset All
Exit
Filter
Prestige
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
Entities
Label
Details
I II III
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia
The Dark Knight (film)
dbr:The_Dark_Knight_(film) • dbpedia
Batman Begins
dbr:Batman_Begins • dbpedia

Figure 1.1: 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 .
}
```

Figure 1.2: The formulated SPARQL example of the running example

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 1.1(right)), as well as the SPARQL query, as shown in Figure 1.2. If the user is not interested in the involvement of the actor "Christian Bale" the constraint "dbo: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.

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.

Although there are some works that aim at offering a QBE-like interaction over Knowledge Graphs in RDF, i.e. [14, 12, 1], our approach has some distinctive characteristics. In particular, "Qbees" [14] neither supports negative examples nor produces a pure SPARQL query, "Query from examples" [12] requires from the user to answer a number of questions, while "Reverse engineering SPARQL queries" [1] cannot receive feedback on the generated constraints.

In the rest of this thesis is organized as follows: Section 2 describes the context and related work, Section 3 introduces the method and the corresponding algorithms. Section 4 describes implementation and evaluation. Finally Section 5 concludes the thesis and identifies issues for further research.

Publications related to this thesis:

- A. Akritidis and Y. Tzitzikas, Demonstrating Interactive SPARQL Formulation through Positive and Negative Examples and Feedback, 26th International Conference on Extending Database Technology, EDBT 2023, March 2023

Chapter 2

Context and Related Work

In this chapter: §2.1 describes the required context, §2.2 discusses the placement of this work in the spectrum of access methods over RDF, §2.3 discusses SPARQL formulators, §2.5 describes example-based SPARQL works while §2.4 discusses cases from different domains where the notion of "example" is used.

2.1 Knowledge Graphs in RDF

A Knowledge Graph (KG) is a structured representation of knowledge that models entities with their properties and relationships using a graph. The entities are represented as nodes with a unique identifier. The relationships are represented as edges connecting two nodes that the relationship applies to. The simplicity of only having two types of elements, nodes and edges, allows for an improved machine interpretability of knowledge.

The Resource Description Framework (RDF) is a standard used mainly to model the data of a KG. The main component of RDF is the triple statement. Each triple is composed of a subject, predicate and object. The subject represents the resource being described, the predicate represents the relationship between the subject and object, and the object represents the value being assigned to the relationship.

The Resource Description Framework Schema (RDFS) is a collection of classes that allow for a more standardised description of the entities and relationships of a KG.

For example for the movie "The Dark Knight" if we want to model some basic properties (type/label) and some relationships with some people (director/actor) we could have:

```
dbr:The_Dark_Knight rdf:type dbpo:Film .
dbr:The_Dark_Knight rdfs:label "The Dark Knight (film)"
dbr:The_Dark_Knight dbo:director dbr:Christopher_Nolan
dbr:The_Dark_Knight dbo:starring dbr:Christian_Bale
dbr:The_Dark_Knight dbo:starring dbr:Michael_Caine
```

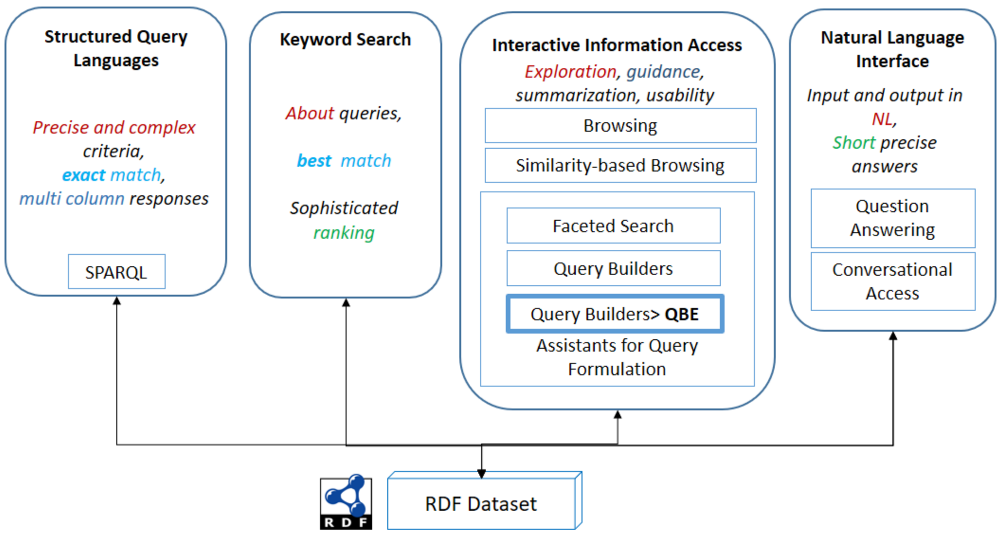

Structured Query Languages
Precise and complex criteria, exact match, multi column responses
SPARQL
Keyword Search
About queries, best match
Sophisticated ranking
Interactive Information Access
Exploration, guidance, summarization, usability
Browsing
Similarity-based Browsing
Faceted Search
Query Builders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

Figure 2.1: An Overview of Access Methods over RDF

Where the prefix for each identifier notates the where it's defined (db stands for dbpedia).

Overall, the use of knowledge graphs has become widespread in various domains, where large amounts of information need to be processed and analysed, including search engines, recommendation systems, natural language processing and others. They occupy the space between knowledge being stored in natural language and table format of the classic relational database. They are more structured than natural language but with less constraints and more flexibility than relational databases.

2.2 Overview of Access Methods over RDF

There are several access methods over RDF, as it can be seen in Fig. 2.1 (extended version of that of [16]). 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. [16]), c) Interactive Information Access that includes plain browsing, similarity-based browsing [2], faceted search [23], and query builders, and d) Natural Language Interfaces (like QA [15]).
Our Placement. Our work falls in the category of Interactive Information Access, specifically on assistants for query formulation.

2.2.1 SPARQL

The SPARQL Protocol and RDF Query Language (recursive acronym SPARQL) is widely used as the main query language for RDF data. Similarly to other query languages, it is designed to provide a way to access and also manipulate data. In this thesis we only consider the access part of the language.

SPARQL supports a wide range of queries from simple filtering and pattern matching to complex aggregate queries. For example, using the KG of "The Dark Knight" from the previous section, if we want to retrieve all the actors taking part in the a movie we could have

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

SELECT ?actor WHERE {
  dbr:The_Dark_Knight dbo:starring ?actor .
}
```

The query uses the SELECT statement to specify that we want to retrieve the values of the variable "?actor". The WHERE statement specifies the pattern that we are searching for. In this case, we are looking for triples where the subject is "The Dark Knight" and the predicate is "starring". The object of these triples will be the names of actors who starred in the movie, represented by the variable "?actor". The results of the query will be a table with a single column for the actor. There are multiple actors in the movie so each actor will appear as a separate row in the table.

More complex patterns are also supported, for example chaining of properties from the object of one triple to the subject of another triple (property paths) can also be modelled:

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

SELECT ?movie WHERE {
  ?movie dbo:director ?director .
  ?director dbo:birthPlace dbr:London .
}
```

Here we want to retrieve all the movies that have a director from London. The variable "?director" is used only internally to create a property path from the variable "?movie" to London.

For a complete deception of RDF and SPARQL see: `https://www.w3.org/TR/rdf11-concepts/` and `https://www.w3.org/TR/sparql11-query/`

2.3 SPARQL Query formulators

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 [13] and YASGUI [19] simply assist the user to write the query with the main interface being a text field for SPARQL. More visual Query builders, like SPARKLIS [7], ResearchSpace [17], Sparnatural [8], and Assistive Query Building (A-QuB) [11], enable the user to construct a query step by step using interactive graphical interfaces. For instance SPARKLIS [7], 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 [17], Sparnatural [8], and Assistive Query Building (A-QuB) [11] 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.

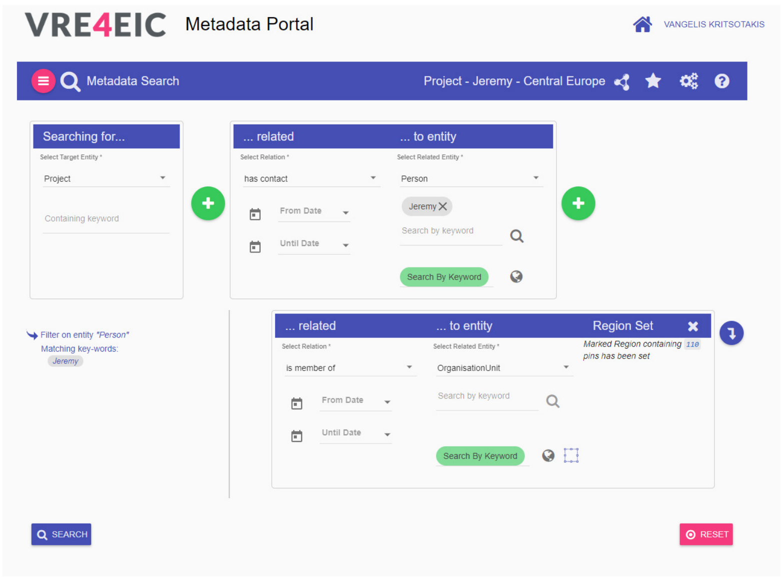


Figure 2.2: A-QuB [11] multiple nested filters

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 [4]), 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 [25] 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 [9] for an overview of older visual query builders published prior to 2015).

Faceted Search systems over RDF (see [23] 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. A recent system for Faceted Search

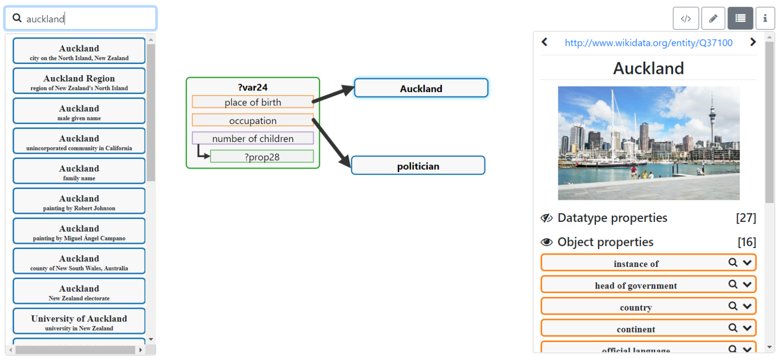


Figure 2.3: RDF Explorer [25] example visual query

over RDF that supports also analytic queries is [18]

Somehow relevant are the extensions of SPARQL with *similarity*, like [26, 6] 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.

2.4 Using the notion of “Example” in other Domains and Tasks

Query by Example in Relational Databases. In the context of relational databases, almost 50 years ago, Moshe M. Zloof developed the Query-by-Example language [28]. 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 [29] [27]. A study of real users (of that era) trying to solve problems using Query-by-Example [22] has shown promising results. However the number of tables/columns of the database was relatively small if we consider the context of knowledge graphs and the number of properties. Also mapping of a knowledge graph to a table/column interface usually leads to small number of tables with very large number of columns which is an undesired structure when a user has any direct interaction with tables. 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 [28] 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* [20] 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 [24], 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.

2.5 Example-based SPARQL works

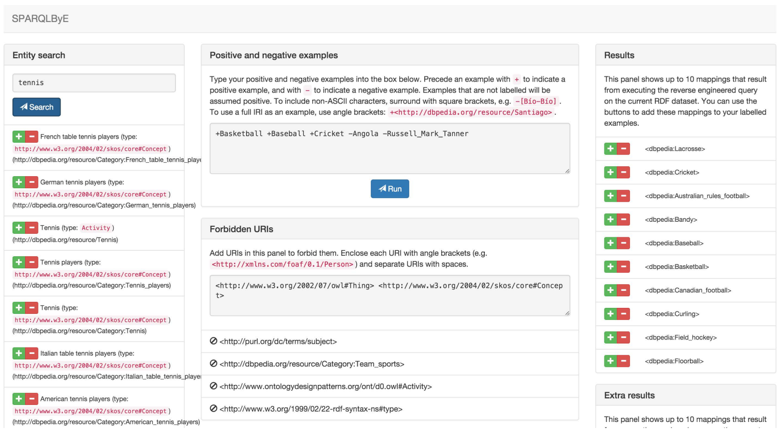
There have been other works where examples are used as the starting point of finding a desired set of results. The term "example" is used in every case but, in the context of each work, the definition of the term is different. In our work, an example can be categorized into four types, which are based on the combination of whether the example has a positive or negative connotation, and whether it refers to a entity or a property-value pair.

In "Query from examples" [12] the user of the system provides examples in a form of a dataset-result pairs 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" [14] 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" [1] and works that use the same technique (eg. SPARQLByE [3]) the examples are positive and negatives entities without 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


SPARQLByE

Entity search

tennis

Search

French table tennis players (type: http://www.w3.org/2004/02/skos/core#Concept) (http://dbpedia.org/resource/Category:French_table_tennis_playe

German tennis players (type: http://www.w3.org/2004/02/skos/core#Concept) (http://dbpedia.org/resource/Category:German_tennis_players)

Tennis (type: Activity) (http://dbpedia.org/resource/Tennis)

Tennis players (type: http://www.w3.org/2004/02/skos/core#Concept) (http://dbpedia.org/resource/Category:Tennis_players)

Tennis (type: http://www.w3.org/2004/02/skos/core#Concept) (http://dbpedia.org/resource/Category:Tennis)

Italian table tennis players (type: http://www.w3.org/2004/02/skos/core#Concept) (http://dbpedia.org/resource/Category:Italian_table_tennis_playe

American tennis players (type: http://www.w3.org/2004/02/skos/core#Concept) (http://dbpedia.org/resource/Category:American_tennis_players)

Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with − to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. -[Bío-Bío]. To use a full IRI as an example, use angle brackets: +<http://dbpedia.org/resource/Santiago>.

```
+Basketball +Baseball +Cricket -Angola -Russell_Mark_Tanner
```

Run

Forbidden URIs

Add URIs in this panel to forbid them. Enclose each URI with angle brackets (e.g. <http://xmlns.com/foaf/0.1/Person>) and separate URIs with spaces.

```
<http://www.w3.org/2002/07/owl#Thing> <http://www.w3.org/2004/02/skos/core#Concep
t>
```

<http://purl.org/dc/terms/subject>

<http://dbpedia.org/resource/Category:Team_sports>

<http://www.ontologydesignpatterns.org/ont/d0.owl#Activity>

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Lacrosse>
<dbpedia:Cricket>
<dbpedia:Australian_rules_football>
<dbpedia:Bandy>
<dbpedia:Baseball>
<dbpedia:Basketball>
<dbpedia:Canadian_football>
<dbpedia:Curling>
<dbpedia:Field_hockey>
<dbpedia:Floorball>

Extra results

This panel shows up to 10 mappings that result

Figure 2.4: SPARQLByE [3] user interface showing the result of adding labelled examples

queries" however in the techniques described the addition of property-value pairs as possible positive or negative examples cannot be integrated. The addition of property-value pair examples, more specifically negative ones, improves greatly the feedback because it replaces the need for the user to find counter-examples. Counter-example are difficult to be found and may not be interpreted by the system as intended.

Chapter 3

SPARQL by Example: Problem Statement, Principles and Algorithms

3.1 Problem Statement

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 query $ans(q) \supset E_P$ and $E_N \cap ans(q) = \emptyset$. We limit all queries to simple conjunction of constraints represented as the set of those constraints and every constrain c as 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 them with more *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 understands why they get more entities). Therefore, 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 the sets of positive and negative constrain examples, C_P and C_N respectively (where where $C_P \cap C_N = \emptyset$).

As regards negative feedback, the entities in E_N are used to exclude combinations of common constraints. While some of the constraints of the final query q may be met by entities of E_N, the conjunction of all the constraints must reject all the entities of E_N.

So the system can be described as single pure function:

$$\text{SPARQL-QBE}(E_P, E_N, C_P, C_N) = q$$

3.2 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

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

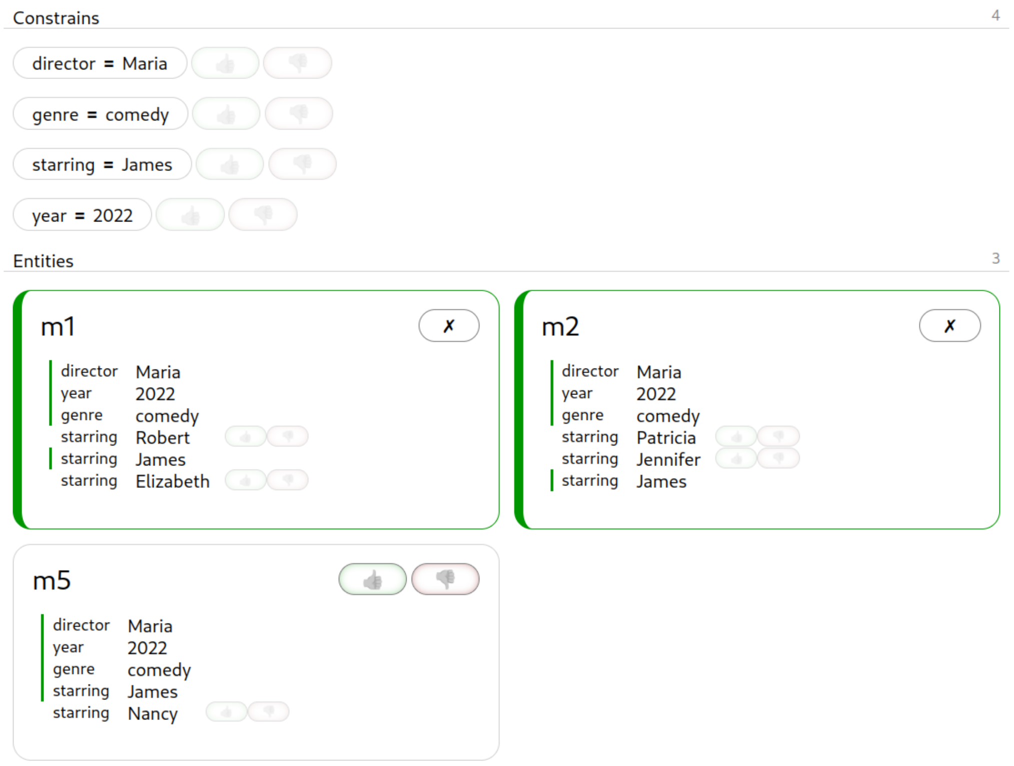
In the following sections, every step and component of the algorithm are described in detail.

3.3 The Exact Process of SPARQL-QBE

Positive Examples. Let K be the set of 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:

$$\mathcal{CC}(E_P) = \{\ (p, v) \mid \forall e \in E_P, (e, p, v) \in K\}$$

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 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 Figure 3.1.



Constrains 4

director = Maria

genre = comedy

starring = James

year = 2022

Entities 3

m1 ✗

director	Maria
year	2022
genre	comedy
starring	Robert
starring	James
starring	Elizabeth

m2 ✗

director	Maria
year	2022
genre	comedy
starring	Patricia
starring	Jennifer
starring	James

m5

director	Maria
year	2022
genre	comedy
starring	James
starring	Nancy

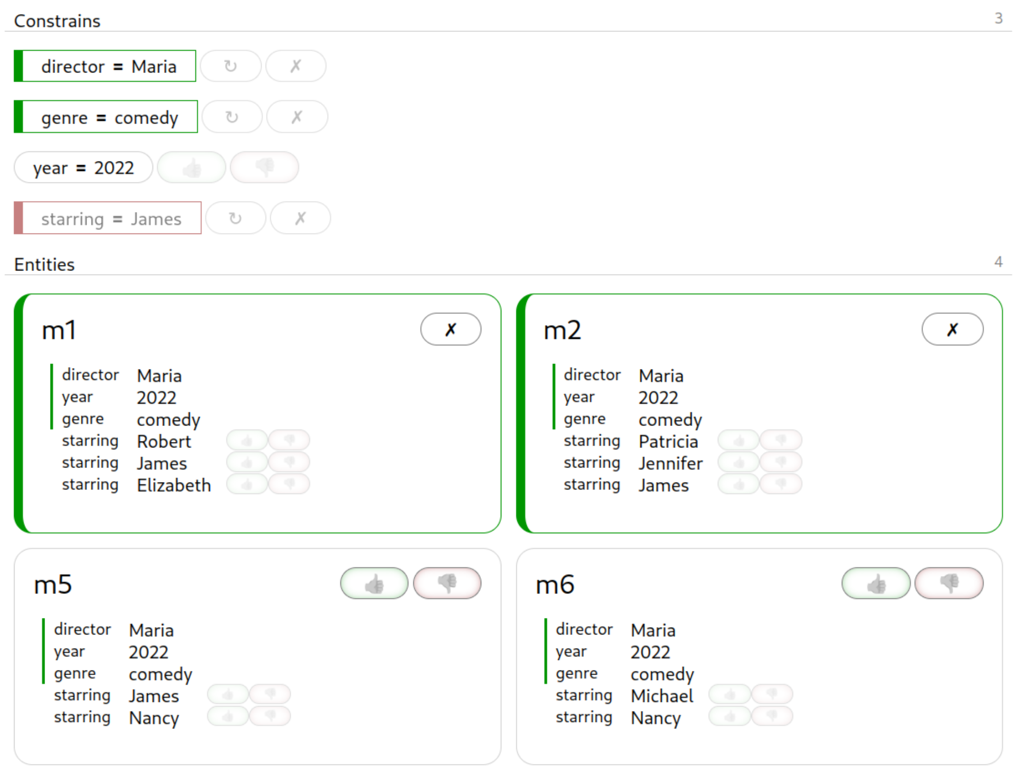
Figure 3.1: $E_P = \{m1, m2\}$ and $ans(q) = \{m1, m2, m5\}$

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:

$$\mathcal{D}(ans(q_0)) = \{\ (p, v) \mid (s, p, v)\ \forall s \in ans(q_0)\ \}$$

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:
$CC' = \{(director, Maria), (genre, comedy), (year, 2022)\}$, and this is the new query, i.e. $q_1 = CC'$ as shown in Figure 3.2.



Constrains 3

director = Maria ↻ ✗

genre = comedy ↻ ✗

year = 2022

starring = James ↻ ✗

Entities 4

m1 ✗

director	Maria
year	2022
genre	comedy
starring	Robert
starring	James
starring	Elizabeth

m2 ✗

director	Maria
year	2022
genre	comedy
starring	Patricia
starring	Jennifer
starring	James

m5

director	Maria
year	2022
genre	comedy
starring	James
starring	Nancy

m6

director	Maria
year	2022
genre	comedy
starring	Michael
starring	Nancy

Figure 3.2: $E_P = \{m1, m2\}$, $C_P = \{(director, Maria), (genre, comedy)\}$, $C_N = \{(starring, James)\}$ and $ans(q) = \{m1, m2, m5, m6\}$

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. 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 also.

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 the 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:

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

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:

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

From these two possible queries, we can use 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):

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

3.3.1 Ranking

As regards the last step, i.e. *ranking*, one approach is to select the query *with the most constraints*, thus we could write: $q = \arg_q \max_{q \in Q}(|q|)$. 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):

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

We can define the *frequency* of a constraint $c = (pr, vl)$ as

$$freq((pr, vl)) = \frac{(pr, vl)^i}{pr^i}$$

Table 3.1: 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	'dbo:starring'	'dbr:Christian_Bale'	49	125130
1026	'dbo:starring'	'dbr:Michael_Caine'	122	125130

. We can use this metric to *rank* each constraint by either promoting the more frequent ones or the more rare ones (or the user could select the desired option at interaction time). In the running example of the introductory section, the frequencies are those shown in Table 3.1. 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 [10]). 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 $|ans(q)|$ 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 $|ans(q)|$, to offer a kind of "guaranteed query relaxation functionality", enabling the user to gradually get more entities. Table 3.2 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 $|C|$, so it can be used only if $|C|$ 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

Table 3.2: The method $\mathcal{M}_{conj}$

$\|ans(q)\|$	q
4	'dbo:starring = Michael Caine, dbo:starring = Christian Bale, dbo:producer = dbr:Emma_Thomas, dbo:director = dbr:Christopher_Nolan, rdf:type = dbo:Film'
7	'dbo: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	'dbo:starring = Christian Bale, rdf:type = dbo:Film'
122	'dbo:starring = Michael Caine, rdf:type = dbo:Film'

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 Policy: As system's 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 the $Ep = \{e\}$, i.e. the user selected a single positive example, the system will consider all the properties of the entity e as the common constrains $CC(\{e\}) = \{\ (p, v) \mid (e, p, v) \in K\ \}$. Some of these constraints will be discarded by the two methods and a query will be generated. With only a single example the query may contain none of the desired constraints but with the use of the negative feedback the user can direct the system to different constraints.

3.4 Extended constraints

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

3.4.1 Operators

In order to extend the expressiveness of the constraints we include the 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.

3.4.2 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. The property path may consist of a single property element, where it fall-backs to the previous behaviour, two or more elements that construct a property path.

We can also extend the possible user feedback on the constraints from only the property-value pairs to property path-value pairs. The number of such pairs may be very large and possibly infinite based on the fact that most knowledge graphs contain property circles. So they cannot be displayed or provided to the user at once.

Chapter 4

Implementation and Evaluation

In this chapter: §4.1 describes the functionality of the proof-of-concept prototype, §4.2 discusses implementation and datasets and their efficiency, §4.3 discusses the expressive power of `SPARQL-QBE`, §4.4 discusses the scalability, §4.5 is about evaluation and applicability, §4.6 provides a comparison with related approaches and systems, §4.7 presents the dataset applicability, §4.8 presents some of the current limitations.

4.1 Functionality 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 4.1.

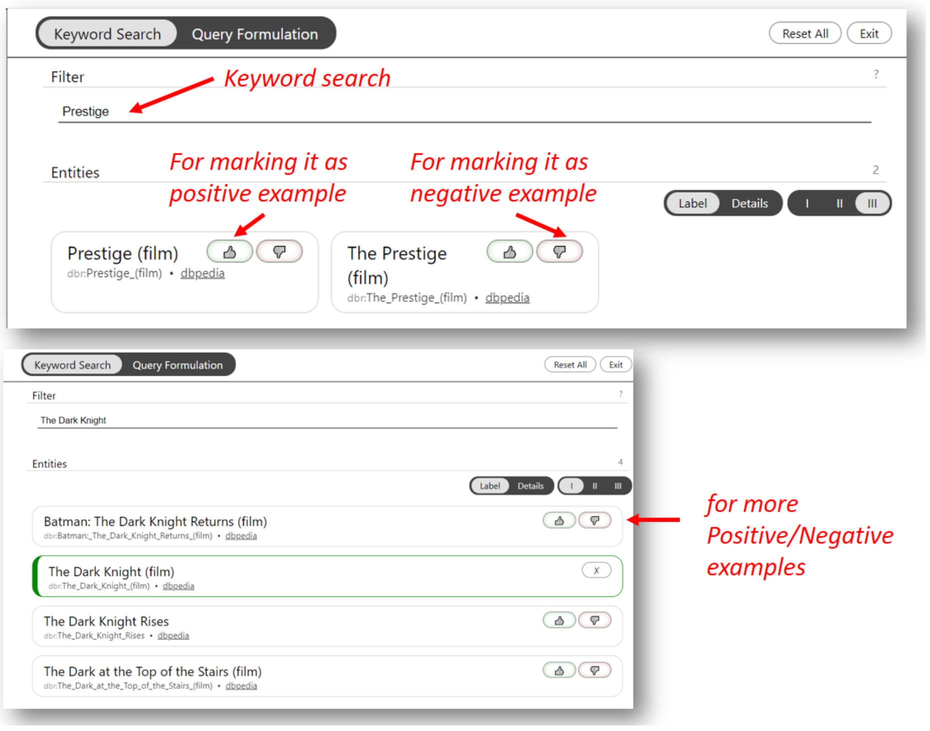
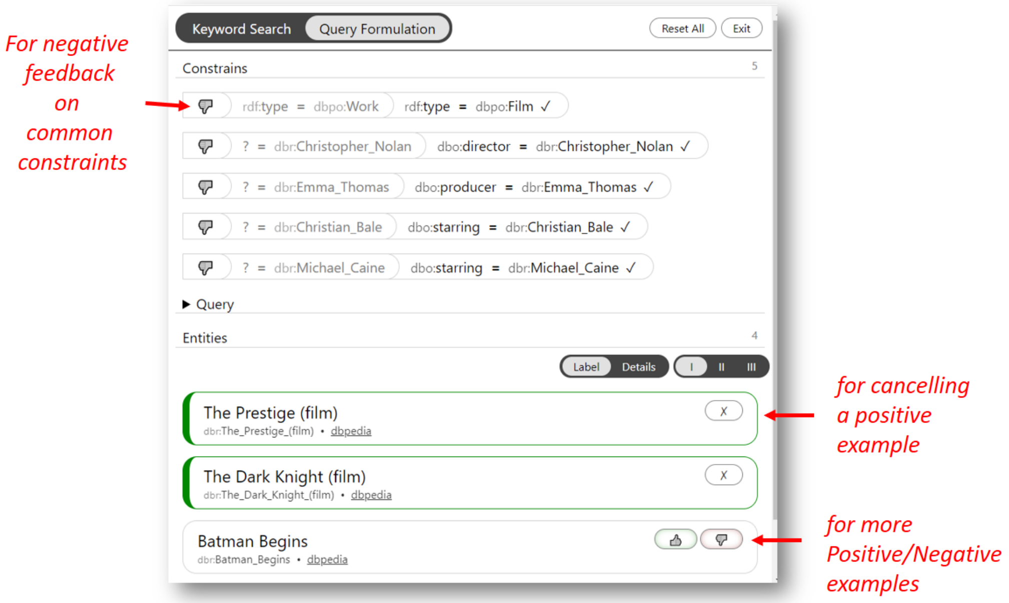

Keyword Search
Query Formulation
Reset All
Exit
Filter
Keyword search
Prestige
Entities
For marking it as positive example
For marking it as negative example
2
Label
Details
I
II
III
Prestige (film)
dbr:Prestige_(film) • dbpedia
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia
Keyword Search
Query 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

Figure 4.1: 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 4.1(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 4.2.


For negative feedback on common constraints
for cancelling a positive example
for more Positive/Negative examples
Keyword Search
Query Formulation
Reset All
Exit
Constrains
5
Query
Entities
4
Label
Details
I
II
III
X

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 ✓

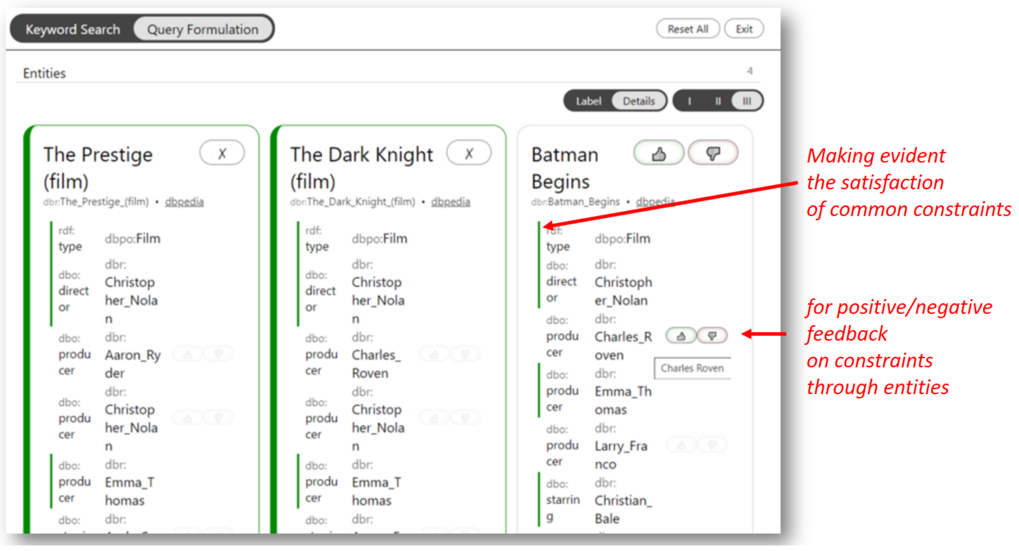
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia

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

Batman Begins
dbr:Batman_Begins • dbpedia

Figure 4.2: SPARQL-QBE: Step 2: Negative feedback on common constraints, and more positive/negative examples

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 4.3.


Making evident the satisfaction of common constraints
for positive/negative feedback on constraints through entities
Keyword Search
Query Formulation
Reset All
Exit
Entities
4
Label
Details
I
II
III
X
Charles Roven

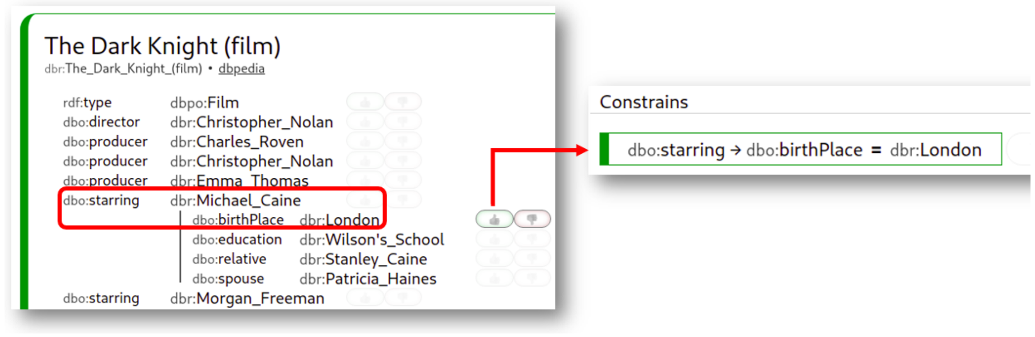
The Prestige (film)
dbr:The_Prestige_(film) • dbpedia
rdf:type dbpo:Film
dbo:director dbr:Christopher_Nolan
dbo:producer dbr:Aaron_Ryder
dbo:producer dbr:Christopher_Nolan
dbo:producer dbr:Emma_Thomas
dbo: dbr:

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: dbr:

Batman Begins
dbr:Batman_Begins • dbpedia
rdf:type dbpo:Film
dbo:director dbr:Christopher_Nolan
dbo:producer dbr:Charles_Roven
dbo:producer dbr:Emma_Thomas
dbo:producer dbr:Larry_Franco
dbo:starring dbr:Christian_Bale

Figure 4.3: SPARQL-QBE: Making evident constraint satisfaction, and constraint feedback through entities

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 4.4. 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.



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

Figure 4.4: The ongoing extension of `SPARQL-QBE` with path expressions

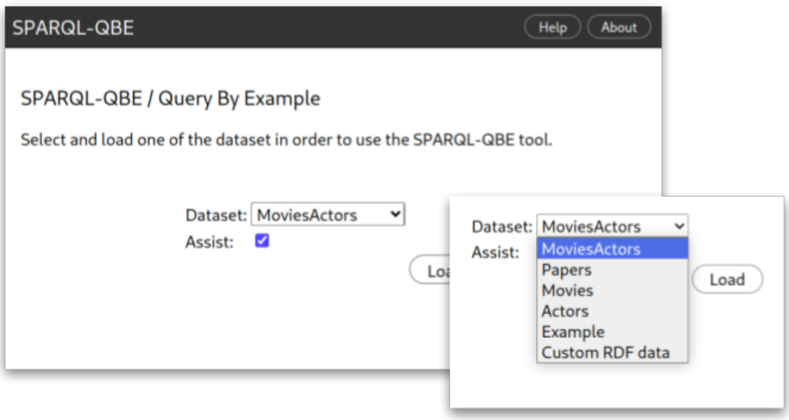
4.2 Implementation

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.

4.2.1 Datasets

We have tested various datasets. For the needs of the task-based evaluation with users we constructed a dataset containing most well documented films and actors from DBpedia whose size is 1,083,029 triples (112,668 films, 43,157 actors). Also we constructed a smaller more inner connected dataset of a subset of published paper and their references from Scopus with size of 532,002 triples (24,431 papers).

A deployment of SPARQL-QBE with these datasets (named MoviesActors and Papers) is accessible through `https://demos.isl.ics.forth.gr/SPARQL-QBE/`.



SPARQL-QBE / Query By Example

Select and load one of the dataset in order to use the SPARQL-QBE tool.

Dataset: MoviesActors

Assist: ✓

Figure 4.5: SPARQL-QBE dataset selection screen

As shown in Figure 4.5 the user has also the ability to load a different dataset by selecting the “Costume RDF data” option and then providing the triples of the dataset in Turtle format.

4.2.2 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 4.6, where we can see the impact of caching for the ranking methods.

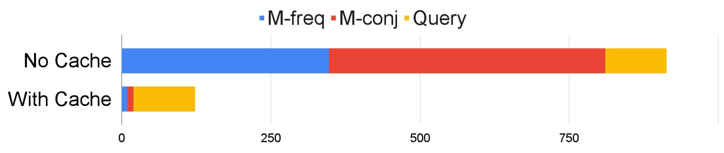


Figure 4.6: Single feedback loop times (with and without caching)

4.3 Expressive Power

The system can formulate single variable single class conjunctive queries.

The system supports different classes however all the examples must be of the same class (or of a common super-class). The class is going to be used as the first generated constraint. The system is designed with the single class query in mind and in the context of examples it would not be clear for a collection of positive examples, where each element has different class, how it should be interpreted.

Except from the class restrictions the rest of the generated queries consist of a set of constrains in conjunction. 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 can also be supported. 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 straight forward visual representation of the query as a single list of constraints. Because the goal is not to generate the perfect query in every feedback loop step, in fact some relaxation is required based on nature of the system, and because a visual representation of a query with multiple combination of conjunctions and disjunctions is difficult to visualize in way that allows for feedback, we consider that the conjunctive only queries are sufficient as a starting point.

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”).

4.4 Scalability

All operation have execution times that are linearly depended on the number of entities and properties given as examples and the SPARQL query execution operation that is provided.

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, 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 with each loop a single query execution is performed. In worst case there will be as many loops 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 can perform an initial reduction of the valid constraints using $\mathcal{M}_{freq}$. By the size of the reduction we can now indirectly provide a upper bound to the generated power set size.

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

Triplestores. For the system to utilize a triplestore instead of the current in-memory dataset model the requirement is query execution endpoint and some capability of keyword search. After the initial keyword search for the first examples the system only requires query executions. Keyword search requests could also be transformed to SPARQL queries in case of no keyword search services.

More specifically all queries, except the for the keyword search, if translated to SPARQL have the following form:

```
SELECT ?x WHERE {
  ?x prop1 value1 .              # prop1 = value1
  ?x prop2 value2 .              # prop2 = value2
  ...
}
```

Each of the property-value pairs represent a single simple constraint. To support the extended constraints with operators and property paths we have to introduce the MINUS and FILTER statement of SPARQL. More specifically, to support the operator $\neq$ they property value pairs are moved inside a MINUS statement and to support the arithmetic operators $<, \leq, \geq, >$ a FILTER statement is generated with the property, value and the operator:

```
SELECT ?x WHERE {
  ?x prop1 value1 .                # prop1 = value1
  MINUS { ?x prop2 value2 . }      # prop2 != value2
  ?x prop3 ?y .                    # prop3 > value 3
  FILTER { ?y > value3 }
  ...
}
```

The two ranking methods need to retrieve the number of entities that meet some constraints. So they require the same queries as shown previously with the only difference being that the results are aggregated with COUNT.

```
SELECT COUNT(DISTINCT ?x) WHERE {
  ...
}
```

Also most of the requests can be cached, especially during a session where the same properties are ranked in every feedback loop.

Lastly for displaying the query results a second series of the queries may be performed to collect all the properties of the referred entities, if they are not already known from previous queries.

4.5 Evaluation

Analytically. In contrast to 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. There are where it is easier for the user to point to examples than to conditions that entities satisfy, eg. a user may point to two songs of a similar style of music without knowing the actual name of that style. Also even if the user can point to the desired constraints the number of them are probably more than two which is the number of examples that could provide instead. Therefore it can be faster (i.e. it may require less actions from the user) than Faceted Search since in the latter the user has to make an action for each constraint. Roughly, the actions required by the query by example are less if $|E_P| < |\text{desired constraints}|$.

Task-based 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 4.1.

Table 4.1: 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 (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 from two examples. T3 requires the user to give a feedback, an instruction to the feedback loop. 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 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 [5], 20 evaluators are enough for

[1]Like the help page of the prototype.

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%**).

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)

4.5.1 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. ∼ 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 4.7, 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.

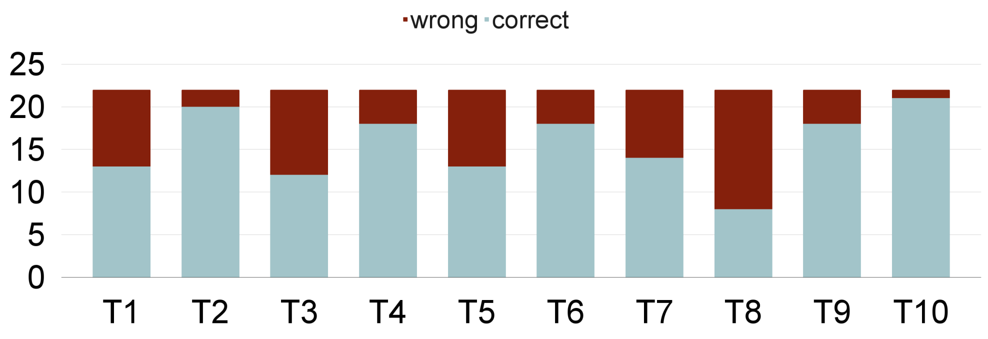


Figure 4.7: Success Rate by Task

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).

4.6 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". 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.

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

Table 4.2: Comparison with example-based systems

	Entity Examples	Negative Examples	Constraints Feedback
Query from examples			
Qbees: query by entity examples	✓		
Reverse engineering SPARQL queries	✓	✓	
`SPARQL-QBE`	✓	✓	✓

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.

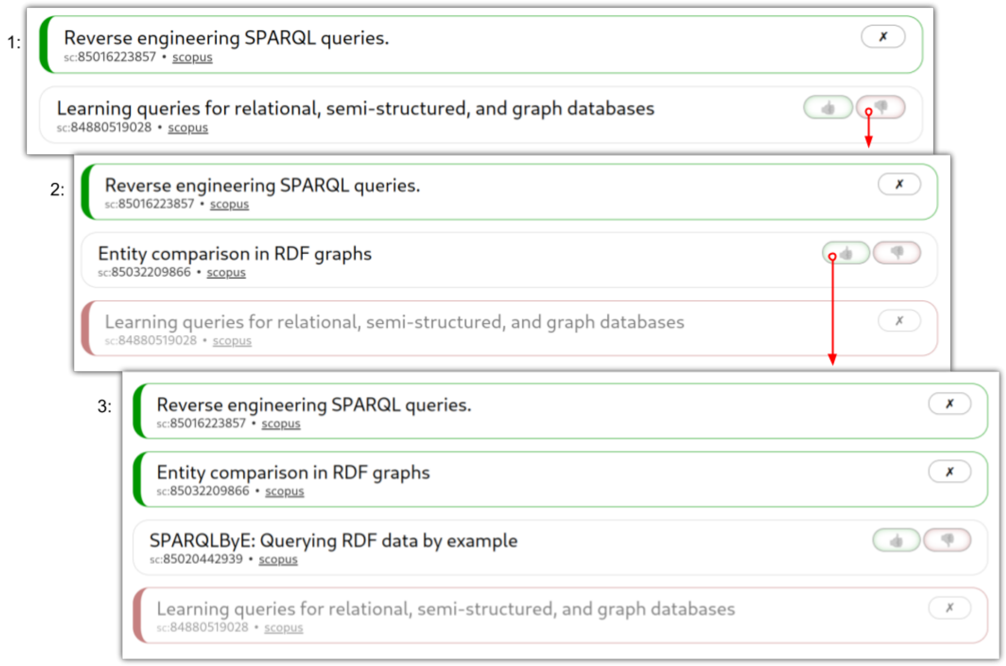
Comparison with systems that support the notion of "Example".
Compared to example-based systems like "Query from examples" [12], instead of explicitly asking for more information from the user before a query is provided to the user, `SPARQL-QBE` provides the best query and then through the feedback loop 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.

As shown in Table 4.2 `SPARQL-QBE` supports both positive and negative examples and also feedback on the generated constraints that is not supported by the other systems.

OPTIONAL. The OPTIONAL operator is used to extract more information from the selected entities. Our generated queries select only the entities. So the OPTIONAL operator never appears at the generated queries. Also the examples given by the user are entities or constraints where there are no null values that could be modeled by the OPTIONAL operator.

4.7 Dataset Applicability

By testing this interaction over various datasets, we have realized that this approach is more efficient that other approaches when the user is not sure about the



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 ✗

Figure 4.8: Discovery scenario with the papers dataset

constraints that should be added to the desired query.

Especially when the commonalities among their desired results are not clear to the user. The reason may be the lack of awareness of the significance of some commonality, eg. a producer of two music tracks, or just be hidden in the large number of properties, eg. a subset of references in two papers. The system will be able to detect and generate the constraints in both cases. The user except from the final query and its results, by inspecting the theses constrains, will be also made aware of these hidden commonalities.

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. 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 with the use of the negative feedback, the system can be used more like a paper discovery system than a query generator. An example scenario, as shown in Figure 4.8:

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

2: The system, based on the feedback, generates a new query that now suggests a 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.

4.8 Applicability and Limitations

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, i.e. the user can start the QBE process by picking entities (e.g. putting them in a cart). Moreover it could complement existing query formulators.

One limitation of the current implementation is that `SPARQL-QBE` will not look for common property paths even though they are supported. The user can provide property path through the interface as we stated but they will never be automatically generated. A second current limitation there is no special handling for numbers/dates and by extension arithmetic relations cannot be utilized.

Chapter 5

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 manner that does not pre-suppose knowledge of the query language or the contents of the Knowledge Graph. The proposed method can exploit (and can be combined with) other access methods (keyword search, browsing), it supports gradual formulation, and various kinds of interactive feedback. We have detailed the algorithmic aspect, 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 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.

5.1 Future Research and Work

There are several issues that are worth further research.

For instance, as described in the limitations, there many possible extensions to the generation of the query constraints. Property path constraints, that are currently only provided by the user, could be generated by the system by performing some traversal through the properties. At a certain point very long paths provide little to no informational value for the initial entity, so for the length of the properties path a limit could be reasonably defined in order to simplify the search. Arithmetic relations could also work well with the current system, but it may be a difficulty detecting such relations. Especially, range constrains for a number or a date would heavily impact the complexity of the generation. However, similarly to the property path, the user could be given the option to manually change a generate equality constraint to an inequality or a range constraint.

More generally some features that also improve the system:

Query editing would allow the user to providing feedback to the system by directly editing the SPARQL of the query. The system would then have to convert the provided SPARQL to the constraints format in order to be able to be passed to the main operation of the query generation. For very complex queries components that cannot be converted to a constraint, they could be handled as a black box, however in that case queries could be generated that contain duplicate or contradictory SPARQL statements.

The results of the query are used to provide feedback to the system, queries with any results other that the entities limit the feedback options of the user. So a third phase could be introduced, after the "find examples" and "provide feedback", where the user has no more feedback to provide and it can provide information about how exactly the results of the query should be "transformed", for instance, some property values instead of the entities or an aggregate of them (eg. count, avg). The fact that the query generated provides a single column of entities would allow for simple interface. A list of properties and possible aggregates and groupings for each would be sufficient for most generated queries.

Finally, for more advanced users, the could allow more parameterization of the internal process. The user could provide the desired size of resulting query results. Based on that the ranking methods and the constraint selection processes would be adjusted accordingly. Also instead of absolute size, the user could feedback by requesting more or less results and the system relatively adjust. With these options then system could also be used to generate a query that describes only the given examples and no other entity. Also the user could provide the importance of the properties in the context of the current query generation in order to improve the ranking methods. The importance could be described as a single value or a comparison between two properties. This process could be very difficult and time consuming, so the system could instead ask for the importance of the properties that would effect the ranking methods based on the given examples.

Bibliography

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

[2] Manos Chatzakis, Michalis Mountantonakis, and Yannis Tzitzikas. RDF-sim: similarity-based browsing over DBpedia using embeddings. *Information*, 12(11):440, 2021.

[3] Gonzalo Diaz, Marcelo Arenas, and Michael Benedikt. Sparqlbye: Querying rdf data by example. *Proceedings of the VLDB Endowment*, 9(13):1533–1536, 2016.

[4] Martin Doerr. The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. *AI magazine*, 24(3):75–75, 2003.

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

[6] Sebastián Ferrada, Benjamin Bustos, and Aidan Hogan. Extending SPARQL with similarity joins. In *International Semantic Web Conference*, pages 201–217. Springer, 2020.

[7] Sébastien Ferré. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web*, 8(3):405–418, 2017.

[8] Thomas Francart. Sparnatural. `https://sparnatural.eu/`, 2021.

[9] Pavel Grafkin, Mikhail Mironov, Michael Fellmann, Birger Lantow, Kurt Sandkuhl, and Alexander V Smirnov. SPARQL query builders: Overview and comparison. In *BIR Workshops*, pages 255–274. Citeseer, 2016.

[10] Yannis Ioannidis. The history of histograms (abridged). In *Proceedings 2003 VLDB Conference*, pages 19–30. Elsevier, 2003.

[11] Vangelis Kritsotakis, Yannis Roussakis, Theodore Patkos, and Maria Theodoridou. Assistive query building for semantic data. In *SEMANTICS Posters&Demos*, 2018.

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

[13] Luke McCarthy, Ben Vandervalk, and Mark Wilkinson. Sparql assist language-neutral query composer. *BMC bioinformatics*, 13(1):1–9, 2012.

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

[15] Christos Nikas, Pavlos Fafalios, and Yannis Tzitzikas. 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*. Springer, 2021.

[16] Christos Nikas, Giorgos Kadilierakis, Pavlos Fafalios, and Yannis Tzitzikas. Keyword search over rdf: Is a single perspective enough? *Big Data and Cognitive Computing*, 4(3):22, 2020.

[17] Dominic Oldman and Diana Tanase. Reshaping the knowledge graph by connecting researchers, data and practices in ResearchSpace. In *International Semantic Web Conference*, pages 325–340. Springer, 2018.

[18] Maria Evaggelia Papadaki and Yannis Tzitzikas. Rdf-analytics: Interactive analytics over rdf knowledge graphs. 2023.

[19] Laurens Rietveld and Rinke Hoekstra. The yasgui family of sparql clients 1. *Semantic Web*, 8(3):373–383, 2017.

[20] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Journal of the American society for information science*, 41(4):288–297, 1990.

[21] Amit Singhal et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[22] John C. Thomas and John D. Gould. A psychological study of query by example. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, AFIPS '75, page 439–445, New York, NY, USA, 1975. Association for Computing Machinery.

[23] Yannis Tzitzikas, Nikos Manolis, and Panagiotis Papadakos. Faceted exploration of rdf/s datasets: a survey. *Journal of Intelligent Information Systems*, 48(2):329–364, 2017.

[24] Yannis Tzitzikas and Carlo Meghini. Ostensive automatic schema mapping for taxonomy-based peer-to-peer systems. In *International Workshop on Cooperative Information Agents*, pages 78–92. Springer, 2003.

[25] Hernán Vargas, Carlos Buil Aranda, and Aidan Hogan. RDF Explorer: A visual query builder for semantic web knowledge graphs. In *ISWC Satellites*, pages 229–232, 2019.

[26] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. Semantic SPARQL similarity search over rdf knowledge graphs. *Proceedings of the VLDB Endowment*, 9(11):840–851, 2016.

[27] M. M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.

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

[29] Moshé M. Zloof. Query-by-example: The invocation and definition of tables and forms. In *Proceedings of the 1st International Conference on Very Large Data Bases*, VLDB '75, page 1–24, New York, NY, USA, 1975. Association for Computing Machinery.