Reverse Engineering of SPARQL Queries using Examples

Anonymous
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623

Abstract—Semantic Web offers a great variety of public datasets for use to end users but the users who are unaware of the Semantic Web technologies such as RDF and SPARQL query language will face obstacles in making complete use of the data. SPARQLByE deals with these issues by letting the users query the data with examples and thus reverse-engineers SPARQL queries. The paper first provides a brief introduction to the problem, related work. In the subsequent section, detailed implementation details about SPARQLByE and the main components which perform reverse engineering of the query are provided. The paper illustrates how SPARQLByE guides the users in understanding the structure of data and developing insights from it.

Index Terms—Reverse Engineering, SPARQL, Semantic Web, RDF triples.

I. INTRODUCTION

In current times, there are multiple publicly available data sets, these data sets can be used effectively if users can query them property to develop insights from the data. For Naive users, who have limited knowledge of SQL or Relational Algebra this can prove to be a challenging task [1]. The paper Reverse Engineering SPJ-Query from Examples [1] studies learning i.e. reverse engineering select-project-join query satisfying a given set of positive and negative examples. The system aims at generating all queries for a sample data and the naive users can analyze the tuples as positive and negative and find the tuples of interest. In [1] a user example set is used as reference which contains positive and negative tuples and the goal is to determine if there exists a query to retrieve all the positive tuples. The topic of Reverse Engineering has been studied since the past decade, few examples include the XPath from XML examples in databases domain, learning Tree patterns from Graph examples and path queries from Graphs [1]. The paper Reverse Engineering with SPARQL Queries [2] talks about Reverse Engineering specifically with SPARQL, a query language for information retrieval for the Semantic Web. Semantic Web provides framework, interfaces for data to be used and shared across applications and enterprises [3]. The main idea of the project is Query Reverse Engineering i.e. to re-generate the query that produced a given output table from a given database. The combined idea of paper [1] and [2] is to form a query based on positive and negative examples from the sample data and [1] talks this aspect with respect to SPARQL.

II. RELATED WORK

Semantic Web consists of standards that promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF). SPARQL is a query language used to manipulate, modify RDF data.

The approach of Reverse Engineering has been heavily used in a Relational Database setting. The main idea of Query By Example [4] is, to construct an alternative Query i.e. if we have the output of query Q on a Database D i.e. Q(D), we have to construct an alternative query Q ’ such that Q(D) and Q ’(D) are instance equivalent. To do this, a data classification technique called at least one semantics is devised. A new dynamic class labeling technique is introduced and provides several optimization techniques to efficiently compute Instance Equivalent Classes (IEQs). The idea is to find queries which provide the similar output and the number of same tuples they contain. The paper takes into consideration 3 datasets and talks about a Tree based Classifier with At least One Semantics (TALOS). The Semantics provide the relationship between the Rel (Q), Sel (Q) of a Relational Database Query and how to efficiently assign class labels for computation of IEQs and generating other Queries on having provided the original Query.

Query Reverse Engineering [4] is a significant extension that generalizes QBO along three key dimensions of the problem space: (1) The original query Q is unknown. (2) The derived query Q ’ belongs to a more expressive query fragment beyond the simple select-project-join (SPJ) query fragment. (3) The database D has multiple versions. Here, the Original Query doesn’t exist. This paper is related to [1] as many aspects discussed are w.r.t. SPJ.

Paper Query Semantic Web Data with SPARQL [5] and Query By Output [6] lays foundation for the work described in [3]. This paper talks in detail about RDF and SPARQL, the syntax, semantics and other details. It provides an analysis about SPARQL being the correct language for representing RDF data.

III. MOTIVATING EXAMPLE

The importance of Reverse Engineering a query can be understood from the following example:
A. Categorizations in Data

Query Reverse Engineering can be useful for deriving alternative characterizations of the data. For Example, suppose we use IMDB data for issuing a query “Select all movies directed by James Cameron”. We will get Titanic and Avatar. An Instance Equivalent Query is “Select all movies which grossed over 2 billion”, this query also return Titanic and Avatar. The Instance Equivalent Query thus provides alternative ways in which Data is categorized and it is easy to understand hidden relationships and develop more insights from the data.

B. Database Security

If several Instance Equivalent Queries point to the same result set and if the data is highly sensitive i.e. passwords, encryption keys etc., then this data must have more security measures and protocols to prevent attack from hackers.

IV. REVERSE ENGINEERING SYSTEM DESIGN

For effective utilization and understanding of the Data Model which uses heavy notation of URIs i.e. Web Identifiers it is necessary that the users find it easy to use and understand which isn’t the case here as there would be a of browsing and navigation to be done for one data at a time. A alternative way, is to use query-by-example where users present examples of what they want, and the system generalizes them [7]. Using Querying by Example does not enforce having complete knowledge of the structure of data and for users unfamiliar with SPARQL also can give example and the system suggests the generalizations.

Consider an Example of DBpedia, which has RDF triples which represent knowledge extracted from Wikipedia. Suppose, a user needs a list of all Spanish speaking countries. The user can use a SPARQL query by using the SPARQL endpoint provided by DBpedia (see: http://dbpedia.org/sparql) but if the user is unfamiliar with SPARQL syntax will not be able to retrieve the required information. Even after knowing the semantics and syntax of SPARQL, an in depth knowledge of DBpedia Ontology is required. This issue can eliminated by using Query by Example where the user can specify some positive and negative examples and based on these the system formulates a SPARQL query desired by the user.

A. SPARQLByE System Overview

SPARQLByE is a system for querying Semantic Web data by example [7] which allows users to obtain information without prior knowledge of SPARQL. SPARQLByE has a Reverse Engineering Component which abstracts the user examples into SPARQL query which is evaluated to present answers to the users [7]. SPARQLByE can be used with any RDF Dataset. The input to SPARQL are positive and negative example which are termed as annotated mappings similar to a tuple in a relational database. A SPARQL query Q, when evaluated on a RDF dataset D, returns a set of mappings Q(D) which is the result set of Q on D. A result set implicitly defines an annotated mapping set where the mappings in Q(D) are positive examples and the mappings outside of Q(D) are the negative examples [7]. The SPARQLByE system makes use of OPTIONAL and AND operators. This system is split into two main components which is, the Reverse Engineering Module and the Example Refinement Module. The two modules themselves depend on several lower level functionalities and modules which are elaborated in the following sections. Also, during execution there is a need for executing auxiliary SPARQL queries generated which is done by the local or the public SPARQL end point. The SPARQL System Architecture is as shown in Figure(1).

1) Reverse Engineering Module: SPARQLByE’s Reverse Engineering Module is based on the Reverse Engineering Algorithms mentioned in [2]. The SPARQL queries should be consistent with positive and negative examples and the domains of the mappings should have a tree like structure which is provided by the AOTree structure, in which a node has a parent, may have multiple children along with information about the list of Triples and the desired variables required. The aim of the algorithm is to start reverse engineering based on smaller examples and forming auxiliary queries and moving up to combine and form the resultant query. At each inductive step, it looks to put together a set of patterns which are satisfied by the positive examples (given by AND operator), are not fit by the negative examples and which are consistent with the absent variables (NULL values) present in positive examples [7]. The bottom up approach follows a Greedy approach for building the safe set of patterns. The algorithm only supports full negative examples as opposed to partial negative examples (a negative example with NULLs) as it is not very intuitive
for the users. The main components which take in the positive and negative examples are as follows:

1) LearnerController: It is responsible for maintaining the update functionality of the Observer and to trigger the execution of the Reverse Engineering functionalities. It parses the input given by the user in the form of positive and negative examples and transcribes them into their respective URI to separate them as positive bindings and negative bindings respectively. Along with the positive and negative examples, it parses the forbidden URIs as bad URIs and passes all these on for learning to LearnerDirector.

2) LearnerDirector: Finds out the depth of the positive solutions tree formed and has a filter on only accepting a few negative solutions. Once the initial analysis of depth is done a function learnTreeLike() is used for building the AOTMinTree using the buildMinTree() from BuildMinTree module. When the MinTree is generated, it is checked for negative examples for validity of solution.

3) PNAndTreeLearner: The MinTree formed is passed for validation. This modules is the one responsible for separation of the triples. This modules iterates over all the \((v,v,c), (v,c,v), (c,v,v), (v,c,c), (c,v,c), (c,c,v)\) types of triples to collect the various associated tags and adds this information to the node formed. The ULearned-QueryChecker is the component which has methods to learn about the node formed and validates the node to build an appropriate query.

2) Example Refinement Module: This module is used by the users to determine if the SPARQL query provides results which were expected by them or are close to what they were expecting. If the user does not find the results satisfactory then the refinement module considers the results as potential negative examples for future considerations. URI Searching based on simple keyword search is a feature of SPARQLByE used for suggesting new positive examples in the current context.

B. Experimental Evaluation with Examples

The data set used here is DBpedia. The SPARQLByE User Interface is as seen in Figure(2). The User enters the examples in the section title Positive and Negative examples. The upper right section contains the results which can be used during the query formation process to add more positive or negative examples or both. In the Forbidden URIs section the User can add URIs which are irrelevant or the URIs to not consider while computing the query. The Reverse Engineered SPARQL Query section contains the Query generated. SPARQLByE evaluation is done with the help of following examples.

**EXAMPLE 1:** A user wants a list of Spanish speaking countries and by using SPARQLByE, the user let’s say enters Chile, Bolivia and Venezuela. The query returned for just these positive examples would be:

```
SELECT * WHERE {
  ?x a <http://dbpedia.org/ontology/Country> ;
  <http://xmlns.com/foaf/0.1/Person> ;
  <http://purl.org/dc/terms/subject>
  db:org/resource/Category:speakingCountriesAndTerritories>
}
```

**Listing 1.** SPARQL query with positive examples

As see above, we get the Query to contain only the Country names, to make the query more specific to our requirement we need to add some negative examples. Angola, Brazil are negative examples which are added from the Results section and another positive example Spain is added similarly. After adding these, the results are refined and the query generated is:

```
SELECT * WHERE {
  ?x a <http://dbpedia.org/ontology/Country> ;
  <http://dbpedia.org/class/yago/WikicatSpanish-speakingCountriesAndTerritories>;
}
```

**Listing 2.** SPARQL query for Spanish speaking counties

The Results section for the above case is as shown in Figure(2). The above query now gives a list of Spanish speaking countries which the user wants which are displayed in the results. We needed the negative example for forming the exact structure of the MinTree to know for filtering out.

**EXAMPLE 2:** A user wants a list of Authors who are from Cambridge, Massachusetts then the user would have to enter positive examples of the Authors from Cambridge and provide negative examples of any Author not from Cambridge. The positive examples entered are Ezra Abbot, Leroy Anderson, Vernon Grant, Susan Howe and the negative examples entered are Lena Chen, Molly Fisk, Henry Clausen. The application generates the following Query:

```
SELECT * WHERE {
  ?x a <http://dbpedia.org/ontology/Person> ;
  <http://purl.org/dc/terms/subject>
  db:org/resource/Category:Writers_from_Cambridge,,Massachusetts>
}
```

**Listing 3.** SPARQL query for Writers from Cambridge

The results are as seen in Figure (3). As seen above, The Query formed has the URI which fetches the tag of ‘Writers from Cambridge, Massachusetts’. The application takes into account the tags i.e. variables while computing the relation between examples. Since, SPARQLByE uses DBpedia’s database which contains URIs of Wikipedia data and Wikipedia has several categories of Authors based on location, gender, country and genre. This makes SPARQLByE to fetch the exact query for retrieving further relevant results.

**EXAMPLE 3:** A user wants a list of American Presidents and enters George Washington, John Adams, Grover Cleveland, Barack Obama as positive examples and Pratibha Patil, Pranab Mukherjee, David Granger, Rajendra Prasad as negative examples. The following Query is returned by the application:

```
SELECT * WHERE {
  ?x a <http://xmlns.com/foaf/0.1/Person> ;
```

Rochester Institute of Technology
Fig. 2. SPARQLByE User Interface - shows the results of positive and negative examples

Listing 4. SPARQL query for American Politicians

```sparql
PREFIX db: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/class/yago/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT *
WHERE {
  ?x a db:org/class/yago/WikicatAmericanPeople;
  a dbo:country/United States ;
  foaf:isMemberOf <http://www.w3.org/2002/07/owl#thing> .
}
```

The results are as seen in Figure (4). As seen above, the query returns both the super class of Politician and subclass of American people. Compared to the previous examples, this example does not give us the specific tag of ‘American Presidents’ as required but the results achieved are mostly American Politicians. Similarly, if the user wants a list of Indian presidents, the positive examples are Pratibha Patil, Pranab Mukherjee, Sarvepalli Radhakrishnan, Rajendra Prasad and the negative examples include David Granger, George Washington, John Adams, Grover Cleveland, Barack Obama. The following query is returned:

Listing 5. SPARQL query for Indian Politicians

```sparql
PREFIX db: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/class/yago/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT *
WHERE {
  ?x a db:org/class/yago/WikicatIndianPeople;
  a foaf:isMemberOf <http://dbpedia.org/resource/Category:Writers_from_Cambridge,_Massachusetts> .
}
```

The results are as seen in Figure (5). The above Query also fails to give the specific tag of ‘Indian Presidents’ but does provide the tags of the examples corresponding to Indian people and holding office i.e. politician.

**Example 4:** A user wants a list of Indian cinema actors. The user enters Deepika Padukone, Ranveer Singh, Preity Zinta as positive examples and Fawad Khan, Gal Gadot and Emma Watson as negative examples. The following query is generated:
We get the URI of Indian people from the query. The results obtained are as soon in Figure(7). From the results, we can now add some more negative examples i.e. Amartya Sen and Salman Rushdie since we specifically need Actors. Adding negative examples will help the system search for more specific tags while forming the AOTMinTree and fetch the desired results by eliminating the irrelevant ones. On adding the negative examples, the query generated is as follows:

```
SELECT * WHERE
{ ?x a <http://dbpedia.org/class/yago/WikicatIndianPeople> ;
  a <http://www.w3.org/2002/07/owl#Thing> ;
  a <http://dbpedia.org/class/yago/Actor109765278> .
}
```

Listing 7. SPARQL query for Indian Actors

From the above query, we get the URIs which correspond to Indian people and Actor. Although, we did not get a specific URI of IndianActor, the results obtained are very accurate and hence the query formed is valid and works well in fetching the relevant results. The results are as shown in Figure (8).

C. SPARQLByE Drawbacks

1) Over-fitting: To address the over-fitting issue of SPARQLByE has a user-based customization to avoid over-fitting and over-generalization. For instance, a common
scenario is that all the positive examples are associated with a concept such as ‘Thing’ (http://www.w3.org/2002/07/owl#Thing) and hence, the resulting reverseengineered query hard codes that URI within it. Users can select a set of ‘forbidden URIs’ (e.g. from the current result set) and the reverse-engineering algorithm will then avoid generating a query containing these [7].

2) NULL values: Consider a user who wishes to obtain a list of authors along with their places of death, the latter being relevant only if they have passed away. If the user uses the positive examples Lewis, Oxford, Tolkien, Dorset and Rowling, NULL, and the negative examples 11th Dalai Lama, Tibet and Abbey Lincoln, Manhattan, The SPARQL query formed is:

```sparql
SELECT * WHERE {
  ?x a <http://xmlns.com/foaf/0.1/Person>
  OPTIONAL {
    ?x a db:org/class/yago/Academician109759069> .
    ?y a db:org/ontology/Place> .
  }
}
```

Listing 8. SPARQL query with OPTIONAL keyword

The above query produces the results as seen in Figure(6). We see that the results only show one Academician and all the places that this person has traveled to or has resided in. The system tries to fetch the first ten results rather than ten best results. The results should be able to provide several authors and the respective places but the SPARQLByE system works on a Greedy algorithm to gather data it fetches the first ten results which it gets by executing the query on the SPARQL end-point. This shows that it fails to provide relevant results when AND clause is present in the reverse-engineered query. There may be cases when the default value is NULL but we do not need an OPTIONAL operator to give results which are not completely relevant. Thus, making it a drawback of this system.

3) Nonconforming/ Malformed URI: As seen from the examples above, the domain of URIs being fetched is varied. Also, the URIs may or may not be completely updated to provide the accurate results. In some cases, where data is migrated somewhere else and due to this,
Fig. 8. Results for List of Indian Actors

A valid URI cannot be formed or fetched for a certain case when the system gives an error saying that the Node of MinTree being formed is Malformed and the ULearnedQueryChecker module which is a part of PNAndTreeLearner responsible for validating the SPARQL query formed generates an error leading to an output of ‘NOTHING LEARNED’. Such a case also arises due to multiple open source databases having conflicting information or URI for an example. Hence, even if data is present at a valid URI, the application still outputs ‘NOTHING LEARNED’.

V. CONCLUSION

The main idea of implementation for SPARQLByE is adapted from the SPARQLByE GitHub [8] As seen above, the SPARQLByE system demonstrates Querying RDF data with the help of ‘examples’ which eliminates the need to know SPARQL syntax. The system supports multiple browsers but concurrent querying is not always effective. The number of positive and negative examples provided also has a significant effect on the refinement of the Query. Two positive and at least one negative example is needed to form a meaningful query. The more specific positive and negative examples better is the Query generated. Also, the order in which positive and negative examples are given does not affect the performance but a minimum of two positive and at least one negative example is required for generating some results.

On considering the evaluation, we can see that the results achieved by the system are fairly accurate and relevant to the set of examples provided. The important issue to be considered is the complexity of the algorithm which takes a significant amount of time to compute and generate the Query. This can be improved by generating a single candidate query and checking its correctness via a call to SPARQL query engine [2].

VI. ACKNOWLEDGEMENT

I have taken several efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Dr. Carlos Rivero, Dr. Joe Geigel for their guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project. I would like to express my gratitude towards my parents and member of Rochester Institute of Technology for their kind co-operation and encouragement which helped me in the completion of this project. My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

REFERENCES


