QuestPro: Queries in SPARQL Through Provenance

Efrat Abramovitz
Tel Aviv University
abramovitz2@mail.tau.ac.il

Daniel Deutch
Tel Aviv University
danielde@post.tau.ac.il

Amir Gilad
Tel Aviv University
amirgilad@mail.tau.ac.il

ABSTRACT

We propose to demonstrate `QuestPro`, a prototype interactive system aimed at allowing non-expert users to specify SPARQL queries. Notably, `QuestPro` makes an extensive use of provenance in deriving the SPARQL queries, in two ways. First, we ask users to provide example output nodes *along with explanations* that are then treated as the provenance of the underlying query, guiding the system's search for a fitting query. We have designed an intuitive interface through which users can gradually build their explanations while understanding the connections between the different objects. The system then generates a set of candidate queries and uses provenance to explain each candidate, prompting user feedback to choose between them. We will demonstrate the usability of `QuestPro` using an ontology of academic publications, engaging the audience in the interactive process while explaining the under-the-hood model and algorithms.

PVLDB Reference Format:
Efrat Abramovitz, Daniel Deutch, Amir Gilad. QuestPro: Queries in SPARQL Through Provenance. *PVLDB*, 11 (12): 1994-1997, 2018.
DOI: https://doi.org/10.14778/3229863.3236243

1. INTRODUCTION

Assisting non-expert users in the formulation of database queries has been a goal of multiple lines of work. A prominent approach in this respect is query-by-example (e.g., [3, 2]) which entails a user specifying examples of output that she would like to get from the query, and a system trying to automatically infer the intended query from these examples. Using this approach was shown to be useful when the full output of the query is available or when users are able to provide a large number of representative examples. But coming up with such a set of examples is non-trivial, and unless this is the case, the system may be unable to distinguish the true intention of the user from other qualifying queries. As an example, consider an ontology of authors and papers, and a query asking for all authors with Erdős number 2. Examples for the query output, i.e. example authors, will likely be un-indicative of the actual intended query, since the authors may share many other characteristics such as field of study and alma mater.

We propose to demonstrate a novel framework [1], that uses *provenance* [5, 7, 6] for the inference of SPARQL queries. The high-level idea is to leverage provenance information in two ways: reducing the search space to several candidate queries, and assisting users in choosing amongst them [1].

Users first input examples of nodes they expect in the output of their intended query, and further formulate explanations for these results. The explanations are formulated through our interface, showing the ontology as a graph and allowing users to gradually build their explanations in an intuitive way. Users start from the node they chose as an output example and then choose which adjacent nodes and edges to include in its explanation, expanding the explanation graph in each step. Thus, each explanation is, in fact, a provenance graph with a special node marked as the output. Continuing our example, the provenance of an example author with respect to the intended query will be one of her co-authorship paths to Erdős of length 2.

The system then compiles the output examples and explanations into formal SPARQL provenance and infers candidate queries (which may include union and disequalities). We look for queries who have a mapping to each of the provenance graphs that does not only yield the output node, but also gives the same exact provenance specified by the user. However, this desideratum is not sufficient as there are still many queries that fit this description, and they can also be very general. To further focus our search on relevant queries we attempt to find ones with a minimum number of variables. Intuitively there is a correlation between the number of variables in the query and the "tightness" of its fit to the given examples. Since finding a candidate query with minimum variables is NP-hard, our algorithm infers and ranks k candidate queries according to the number of unions and the number of variables they include, preferring queries which have a low number of both. This procedure clearly focuses the search for a query, but we still have to choose the intended query from multiple candidates. In our example, another candidate query may be one that uses a constant for the name of the third author in the chain if this name was the same in all given examples.

[1]Large parts of our technical description are taken from our full paper [1], where further details may be found.

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3236243

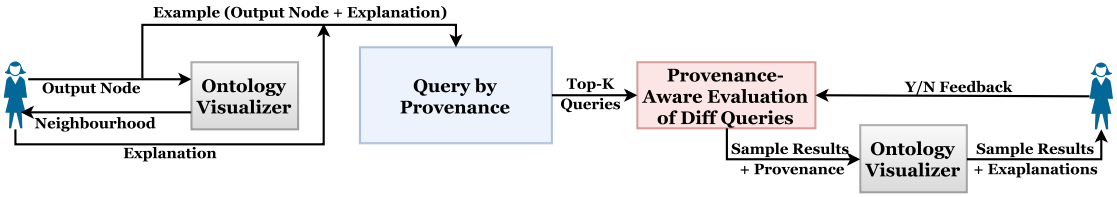


Figure 1: System Architecture

To this end, we also employ provenance in a second manner. After choosing the k candidate queries that fit the description of the user, we utilize provenance to allow unexperienced users to differentiate between the candidate queries, and choose a single one. In our example case, showing not only an example result in the difference between the query results (i.e. an author with Erdős number 2 through a different "middle" author), but rather the provenance of such a result, i.e., the chain going through the other middle author, will allow users to distinguish between the queries.

We will demonstrate `QuestPro` using the SP2B database (a DBLP-based ontology) and allow participants to pose examples of outputs and formulate explanations for them through an intuitive GUI (see Figure 2a). The system will then compute k candidate queries for users to choose from. Users will be presented with examples of results and their explanations to assist them in focusing on a single query. The explanations again will be shown in graph form, visualizing the connection between the different objects in the ontology (see Figure 2b). Once the users have chosen the query, it will be evaluated on the ontology and the results returned to them. In the demonstration, we will show that (1) non-expert users are able to formulate explanations for output examples through our GUI and graphic representation of the ontology, (2) the system is able to infer reasonable queries from just a few output examples and their explanations, and (3) users can provide meaningful feedback and distinguish between queries based on the differentiating results and explanations.

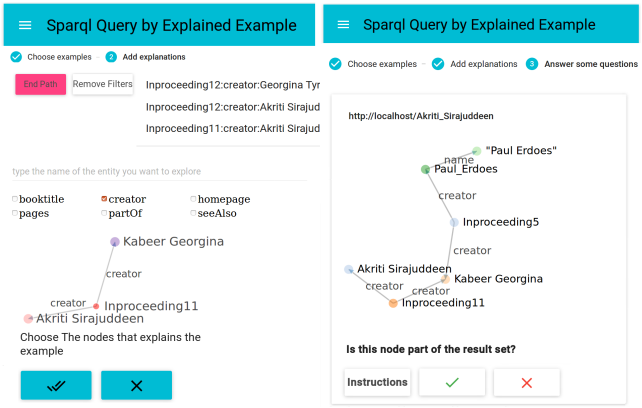
Related Work. There is a large body of literature on learning queries from examples, in different variants, specifically in the context of *data exploration* (e.g., [8, 3]), and SPARQL [2]. The fundamental difference between our work and previous work in this area is the assumed input. Our short demo paper [4] presented a system for relational databases that allows for the inference of a single CQ as opposed to several candidate UCQs with disequalities. Furthermore, the system in [4] is not interactive, while here, due to the inference of several queries, interactive user feedback is crucial. In addition, multiple lines of work have proposed provenance models for SPARQL (e.g., [9, 5]). We have focused on learning queries from explanations based on graph provenance for SPARQL queries rather than the relational model.

2. MODEL AND SYSTEM DESCRIPTION

We next explain and exemplify each component of `QuestPro` (the architecture is depicted in Figure 1) starting with our model for provenance and queries.

2.1 Model

First we define the provenance model and the formal notion of a query that is consistent with a given provenance



(a) Explanation Screen (b) Feedback Screen

Figure 2: Interface Screens

information. We focus on a simple class of SPARQL queries, namely basic graph patterns with a single output node and union thereof, possibly with disequalities.

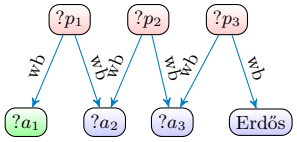


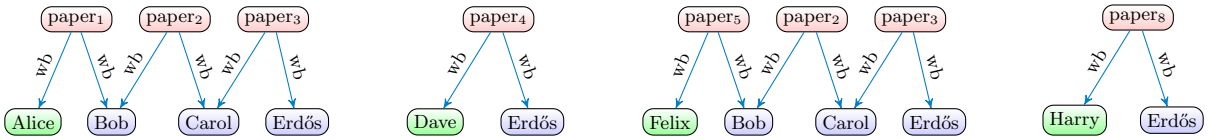
Figure 3: Q_1

EXAMPLE 2.1. *Q_1, Q_3, Q_4 in Figures 3, 5a, 5b are examples of a simple queries while $Q_2 = Union(\{Q_3, Q_4\})$ is not as it is a union of simple queries. Some nodes are variables, notated by ? (e.g., the nodes $?a_1, ?p_1$), and the green node denotes the output.*

For such queries there is an intuitive notion of provenance for a given result node n: the ontology subgraph that is the image of some homomorphism from the query to the ontology graph, which yields n.

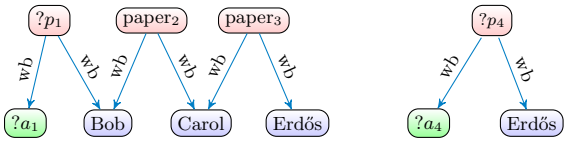
EXAMPLE 2.2. *Consider the query Q_1 depicted in Figure 3 and the ontology subgraph E_2 depicted in Figure 4b. Q_1 matches E_2 since we can define a mapping μ between the vertexes that respects the edges by $\mu(?p_1) = \mu(?p_2) = \mu(?p_3) = paper_4$, $\mu(?a_1) = \mu(?a_2) = \mu(?a_3) = Dave$, $\mu(Erdős) = Erdős$. The output of $Q_1(E_2)$ is the value Dave while the provenance is exactly the graph E_2.*

A query is consistent with a set of examples and their provenance information if evaluating the query yields each of the examples with provenance that is isomorphic to the given one.



(a) E_1 (b) E_2 (c) E_3 (d) E_4

Figure 4: Explanations



(a) Q_3 (b) Q_4

Figure 5: Q_2 is the Union of Q_3, Q_4

Example 2.3. *Consider the four explanations E_1, E_2, E_3, E_4 presented in Figure 4, and their output nodes Alice, Dave, Felix, and Harry. Further consider Q_1 depicted in Figure 3. There is a match μ between Q_1 and E_2 producing Dave whose provenance is exactly E_2 (detailed in Example 2.2). In a similar manner, there is a match between Q_1 and E_1, E_3, E_4, thus Q_1 is consistent with the four explanations.*

2.2 Formulating Explanations

We begin our search for the desired query by using the widespread method of allowing users to provide output examples for the query they would like to run (e.g., [3, 2]), using auto-complete. We then provide a tool that visualizes the ontology as an interactive graph (the "ontology visualizer" component in Figure 1), allowing users to formulate explanations for their output examples (see Figure 2a). As ontologies can be large in size, and may overwhelm the users, we show only the surroundings of each output example, gradually revealing only the necessary parts.

Example 2.4. *Consider a user searching for all authors who have an Erdős number 3. The user gives the authors Alice and Dave as examples and now goes on to formulate the reason she chose these examples. For Dave, the system starts by showing the user all nodes adjacent to Dave in the ontology and she recognizes the paper $paper_4$ written by Erdős, so she chooses this object, revealing the neighboring objects of $paper_4$ which include Erdős. So, she chooses Erdős, resulting in the explanation E_2 depicted in Figure 4b.*

2.3 Query By Provenance

The output objects along with their explanations (compiled by the system to formal SPARQL provenance) are inputted to this component which transforms them into k candidate queries. The algorithms we employ here are detailed in [1] and are based on a greedy heuristic for finding consistent queries with the minimum variables (as this problem is NP-hard). We first describe the algorithm for inferring simple queries and then for inferring unions of simple queries.

Simple Queries. Intuitively, given two explanations E_1 and E_2, at each step, the algorithm decides which two edges from E_1 and E_2 are most "similar", and groups them together. To find which edges to pair, we use a dynamic gain function that is recomputed each time a pair is chosen. The function gets a pair of edges and computes a weighted average considering the number of identical constants in their sources/targets, the number of times they have been paired before with other edges, and the number of times their sources/targets have been paired before. Once all edges are paired, our algorithm builds a query by constructing an edge for each of the pairs. We call this process *merging*.

Example 2.5. *Reconsider the two explanations E_1, E_2 in Figures 4a, 4b. The pair of edges $(paper_3, Erdős)$ and $(paper_4, Erdős)$, has the highest gain since the two edges have the common target Erdős, so this pair is the first to be chosen. Now the gain function is recomputed so the pair $(paper_3, Carol)$ and $(paper_4, Dave)$ has the highest gain (since the sources $(paper_3, paper_4)$ were paired together in the previous step), so the algorithm adds this pair. Continuing on this computation will result in the set of pairs such that all edges in E_1 to the edge $(paper_4, Dave)$ except for the edge $(paper_3, Erdős)$ paired with the edge $(paper_4, Erdős)$. The algorithm assembles the query Q_1 depicted in Figure 3 from the set of pairs.*

In the general case, the input may include more than 2 explanations. In this case, we run the algorithm on each pair of explanations and greedily choose to merge the pair of explanations yielding the maximal gain. We repeat this procedure while there are still explanations to merge, merging not only explanations with other explanations but also explanations with intermediate queries.

Unions of Simple Queries. Now allowing unions, we aim to find a query Q that fits the description given by the user and minimizes a cost function balancing the number of variables and the number of queries in the union: $f(Q) = w_1 \cdot \sum_{q \in Q} |vars(q)| + w_2 \cdot |Q|$ (the weights $w_1, w_2 \in \mathbb{R}$ can be set as we wish).

Example 2.6. *Consider the example $Ex = \{E_1, E_2\}$, where E_1, E_2 are given in Figures 4a, 4b. Also consider the queries $Q = Union(\{E_1, E_2\})$ (i.e. a query with only constants whose projected nodes are the green nodes in the explanations) and Q_1 depicted in Figure 3. Computing the cost function f gives us $f(Q) = w_1 \cdot (0+0) + w_2 \cdot 2$ and $f(Q_1) = w_1 \cdot 6 + w_2 \cdot 1$.*

Our basic algorithm returns a single query by starting from a query which is simply a union of all the explanations given, and merging the two queries/explanations whose merging cost is the smallest, using the algorithm for simple queries. We continue this process as long as there is a merge that decreases the cost function.

We now adapt the algorithm to output k queries. in the first iteration, the procedure chooses the top-k best explanation pairs which yield the simple queries with minimal cost. The output of the procedure is now a list of top-k simple queries, thus outputting k different sets. In every subsequent iteration, the procedure tries to generalize the best

pair from every set, ending up with k^2 sets, and choosing the top-k sets that minimize the cost function.

EXAMPLE 2.7. *Reconsider as input the four explanations in Figure 4 and the weights $w_1 = 1$, $w_2 = 7$ and demonstrate the resulting top-3 queries. After merging in each step the two explanations that most decrease the cost function, we get the top-3 queries Q_1, Q_2, and $Union(\{Q_4, E_1, E_3\})$ seen in Figures 3, 5 and 4.*

2.4 Feedback

We first explain how to add disequalities to a query while maintaining its consistency and then detail how these disequalities assist users to zoom in on a query.

Disequalities. For each of the top ranked queries returned by our algorithm, we may add disequalities between every pair of variables that are mapped to nodes of the same type (this is an additional information in the ontology) but with different values in all explanations

EXAMPLE 2.8. *Reconsider the four explanations E_1, E_2, E_3, E_4 depicted in Figure 4, and the consistent query Q_1 in Figure 3. Consider the variables $?a_1, ?a_2$. when Q_1 is matched with E_2, both variables were assigned to a single constant, so we may not add a disequality. This is the case for all pairs of nodes, and thus Q_1 will also include authors with Erdős number $1, 2$. In contrast, recall query $Q_2 = Union(\{Q_3, Q_4\})$ depicted in Figure 5. Here, the disequalities $?a_1 \neq Bob$, $?a_1 \neq Carol$, $?a_1 \neq Erdős$, $?p_1 \neq paper_2$, $?p_1 \neq paper_3$, and $?a_4 \neq Erdős$ are possible.*

Feedback. To choose a single query out of the candidates, `QuestPro` provides a feedback stage. We start with a collection of k candidate queries, disqualifying one of them at each step. In each iteration, two random candidates are selected Q_1, Q_2 and the query $Q_{diff} = Q_1^{\text{all} \neq} - Q_2^{\text{no} \neq}$ is evaluated and its provenance according to Q_1 is stored ($Q^{\text{all} \neq}$ is Q_1 with all possible disequalities that maintain its consistency and $Q_2^{\text{no} \neq}$ is Q_2 without disequalities). This process produces results that appears in Q_1 and not in Q_2 along with the provenance or explanation according to Q_1. We then ask the user whether a result from this set is relevant, also showing the explanation for this result in graph form according to Q_1 (see Figure 2b). We always prefer results that were in the original input given by the user, as we assume this part of the database is better understood. If the user stated she is interested in the result, we discard Q_2. Otherwise, we discard Q_1.

EXAMPLE 2.9. *Consider the top-3 queries in Example 2.7 as input, i.e., $\mathcal{Q} = \{Q_1, Q_2, Union(\{Q_4, E_1, E_3\})\}$. We begin by evaluating $Q_{diff} = Q_1 - Union(\{Q_4, E_1, E_3\})$, where Q_1 contains all possible disequalities and $Union(\{Q_4, E_1, E_3\})$ contains none. One of the results of Q_{diff} is the author George whose Erdős number is 3, but through an authorship path that does not include Bob and Carol, so we bind this value to Q_1 and evaluate this query resulting in the provenance for George (similar to the explanations in Figures 4a, 4c) stating that its Erdős number is 3. When this result is shown to the user, she responds "yes", i.e. this is a desirable result and explanation, so the query $Union(\{Q_4, E_1, E_3\})$ is discarded. The procedure is performed again with $Q_{diff} = Q_1 - Q_2$ to choose a single query pattern.*

Once we have focused on a query pattern, we can use a slight augmentation of the feedback algorithm, but this time for choosing the right disequalities. We use different "versions" of the chosen query, where each version has a different combination of disequalities.

3. DEMONSTRATION SCENARIO

We will demonstrate that `QuestPro` allows non-expert users to formulate explanations, infer several candidate queries, and provides a useful feedback mechanism for users to select the intended query amongst the candidates. The system will be demonstrated with respect to the SP2B ontology (an ontology containing information about authors, conferences, and papers), through examples such as those presented in this short paper. The demonstration will interactively engage the audience, demonstrating the different facets of the system. For the first part of the demonstration we will use a set of pre-defined output examples and explanations of varying complexity levels. We will first show the audience how to formulate explanations through the ontology visualizer (Fig. 2a), then we will show all candidate queries that have been generated by the system and finally, we will demonstrate the feedback stage where the system depicts both a result and its explanation in an intuitive graph form so that users can understand how this result came to be and whether the explanation fits their intentions (Fig. 2b). We will also show results of a different query without the explanation, to demonstrate the importance of attaching explanations to the results for validation and understandability. We will then allow participants to come up with their own original output examples, formulate explanations for them, and use the system to infer a query. We will continue by showing participants the raw provenance representation, explaining the translation from intuitive graph explanations and highlighting the manner in which `QuestPro` computes queries from these provenance and output examples. Last, we will show how the provenance-aware evaluation of queries is performed efficiently, to allow for interactivity.

Acknowledgments. This research was partially supported by the Israeli Science Foundation (ISF, grant No. 1636/13), and by ICRC - The Blavatnik Interdisciplinary Cyber Research Center. The contribution of Amir Gilad is part of a Ph.D. thesis research conducted at Tel Aviv University.

4. REFERENCES

[1] E. Abramovitz, D. Deutch, and A. Gilad. Interactive inference of sparql queries using provenance. In *ICDE*, 2018.
[2] M. Arenas, G. I. Diaz, and E. V. Kostylev. Reverse engineering SPARQL queries. In *WWW*, 2016.
[3] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive inference of join queries. In *EDBT*, 2014.
[4] D. Deutch and A. Gilad. Qplain: Query by explanation (demo). In *ICDE*, 2016.
[5] F. Geerts, T. Unger, G. Karvounarakis, I. Fundulaki, and V. Christophides. Algebraic structures for capturing the provenance of SPARQL queries. *J. ACM*, 63(1), 2016.
[6] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.
[7] H. Halpin and J. Cheney. Dynamic provenance for SPARQL updates. In *ISWC*, 2014.
[8] T. Sellam and M. L. Kersten. Meet charles, big data query advisor. CIDR, 2013.
[9] Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1), 2011.