

A General-Purpose Visual Query Language for Knowledge Graphs with Bidirectional Transformations

Qiang Fu¹, Xin Wang¹, and Yuan-Fang Li²

¹ College of Intelligence and Computing, Tianjin University, Tianjin, China
{tomqcust,wangx}@tju.edu.cn

² Monash University, Melbourne, Australia
yuanfang.li@monash.edu.

Abstract. In this paper, we present a general-purpose interactive visual query language, GPVQL, to improve the efficiency of end-users' understanding and querying of knowledge graphs. Furthermore, GPVQL realizes the novel capability of flexible bidirectional transformations between query patterns and graph results, therefore significantly assists end-users in formulating queries over large and unfamiliar knowledge graphs in an incremental way. We present the syntax and semantics of GPVQL, discuss our design rationale behind this interactive visual query language, and evaluate the effectiveness of a visual query system based on GPVQL against a number of textual and visual query environments over a large knowledge graph, DBpedia. Our evaluation demonstrates the GPVQL's superiority in effectiveness and accurateness.

Keywords: Knowledge graphs · Visual query language · Interactive · Bidirectional transformation.

1 Introduction

Artificial intelligence has become a powerful tool to meet practical requirements in various domains. Knowledge graphs have been identified as a critical component in diverse AI-based applications. Thus, designing query languages to support the effective and efficient exploration and query answering over knowledge graphs has become a key research problem under insentive investigation. However, a number of key challenges still remain on the generalizability and ease of use of these query languages.

An illustrating example of the process from a question to the corresponding query results is shown in Fig. 1. As can be seen from the textual query (in SPARQL) at the top middle in Fig. 1, it may be difficult for an end-users to quickly learn and use such a query language, or to be familiar with a knowledge graph. Visual query languages can help make it easier for users to construct *query*

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

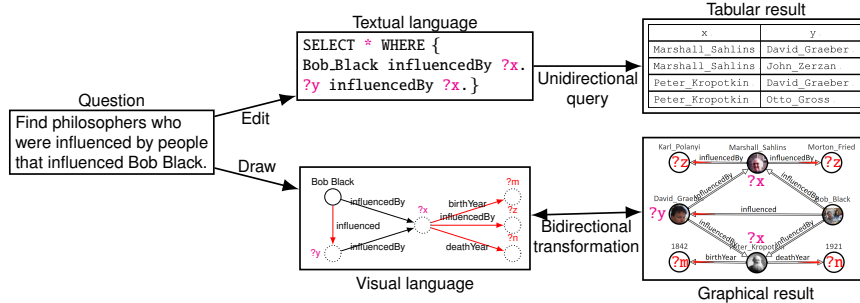


Fig. 1. The process from question to result.

patterns, as shown in the bottom middle of Fig. 1. However, end-users may not understand the correspondence between a query pattern and its results (such as those nodes labeled by the same variable name at the bottom middle/right of Fig. 1) when they are exploring a knowledge graph. Such correspondences allow a user to easily modify and expand the current query to build more complex ones and are supported by the bidirectional transformation functionality provided by our GPVQL visual language. In this paper, we introduce the visual syntax and semantics of GPVQL, discuss our design rationale, and demonstrate its accurateness and effectiveness in a user study.

2 Our Approach

In GPVQL, (1) single circles with dotted and solid lines indicate variables and constants, respectively; (2) two single circles connected by a directed edge denote a basic triple pattern; and (3) double circles and rectangles represent operators and parameters in a query pattern, respectively. We illustrate the syntax and semantics of GPVQL with five examples in Table 1 and the corresponding queries in three textual query languages: SPARQL, Cypher and Gremlin. In GPVQL, end-users do not need to learn a specific textual query language to write queries. What end-users need to learn are just query examples that can then be used in an interactive query-by-example (QBE [2]) way.

Based on our proposed GPVQL language, we have developed a visual interface that addresses these challenges to achieve our goal of enabling end-users to query and explore knowledge graphs. The interface ³ of GPVQL is composed of six main components that are displayed in four panes in Fig. 2. Fig. 2 shows three components: an interactive visual query editor (left pane), keyword and type search panel (right top pane), and query code display panel (right bottom pane). In the top left are buttons for additional functionality: adding new nodes, showing user manual, user study documents, using force-directed layout, etc. In GPVQL, the construction of a query pattern is mainly done in a drag-and-drop manner. The operations (i.e., Expand, Collapse, Filter, Lock, Optional, and U-

Table 1. The examples of visual syntax and semantics in GPVQL.

ID	Query pattern of GPVQL	SPARQL	Cypher	Gremlin
P_1		<code>SELECT ?city WHERE { ?city locatedIn Texas.}</code>	<code>MATCH (x:City)-[:locatedIn]->(y:State) WHERE y.name = "Texas" RETURN x</code>	<code>g.V('Texas') .in('locatedIn')</code>
P_2		<code>SELECT ?city WHERE { ?city metro ?metro.}</code>	<code>MATCH (x:City)-[:metro]->(y:Population) RETURN x, y</code>	<code>g.V() .in('metro')</code>
P_1 AND P_2		<code>SELECT ?city WHERE { ?city locatedIn Texas. ?city metro ?metro.}</code>	<code>MATCH (x:City)-[:locatedIn]->(y:State) WHERE y.name = "Texas" AND (x:City)-[:metro]->(z:Population) RETURN x</code>	<code>g.V('Texas') .in('locatedIn') .and(out('metro'))</code>
P_1 UNION P_2		<code>SELECT ?city WHERE { ?city locatedIn Texas.} UNION {?city metro ?metro.}</code>	<code>MATCH (x1:CITY)-[:locatedIn]->(y:State) WHERE y.name = "Texas" RETURN x1 UNION MATCH (x2:CITY)-[:metro]->(z:Population) RETURN x2</code>	<code>g.V('Texas') .in('locatedIn') .or(out('metro'))</code>
P_1 OPT P_2		<code>SELECT * WHERE { ?city locatedIn Texas. OPTIONAL { ?city metro ?metro.}</code>	<code>MATCH (x:CITY)-[:locatedIn]->(y:State) OPTIONAL MATCH (x:CITY)-[:metroPopul]->(m:Population) WHERE y.name = "Texas" RETURN x, m</code>	<code>g.V('Texas') .in('locatedIn') .or(out('metro'))</code>

nion) are in the right-click context menu. As shown in Fig. 2, GPVQL not only supports keyword queries, but also type-based queries as the starting point. The process starts with a blank visual query editor. A user can choose keyword or type as the starting point for queries. For example, as shown in Fig. 2, after the user adds a new node ①, and input “Bob” as a keyword ②, the related entities will be automatically displayed. Once “Bob Black” is chosen, the thumbnail ③ will be displayed automatically. Further, the user can find people who have in-

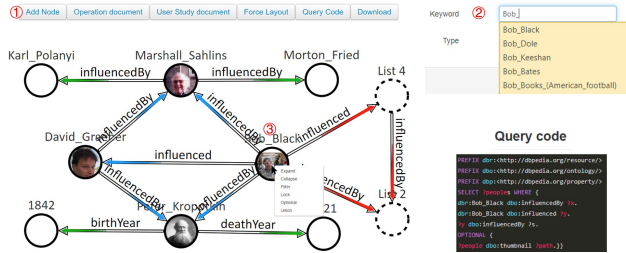




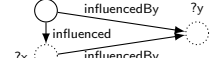
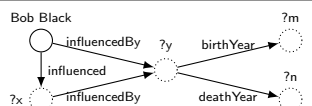
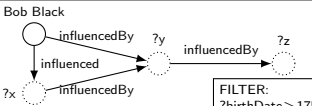
Fig. 2. Example visual query finding people who influenced both Bob Black and people influenced by Bob Black.

fluenced both Bob Black and people influenced by Bob Black (the corresponding query pattern is marked with red arrows). Users can select the Expand operation in the context menu to expand a result. The expanded results are pointed to with blue arrows. When users are interested in some query result, deep exploration to query additional information is supported. For example, users can find the birth year and death year of Peter Kropotkin, or people who influenced Marshall Sahlins (pointed to with green arrows).

3 User Study and Evaluation

We conducted a user study to assess how accurate and effective our new visual query language GPVQL ³ is in comparison to the current semantic query language SPARQL, a visual query language QueryVowl [1], and RDF Explorer [3]. We used the same SPARQL endpoint ⁴ of the DBpedia knowledge graph to ensure the fairness of the study. We created five tasks, shown in Table 2, based on an informal survey of interesting patterns that people formed when exploring prior graph query research works. We recruited 20 participants in total and divided them into four groups evenly. Each group uses a different tool, e.g., Group A + GPVQL, Group B + SPARQL, Group C + QueryVOWL, and Group D + RDF Explorer.

Table 2. Tasks of the user study.

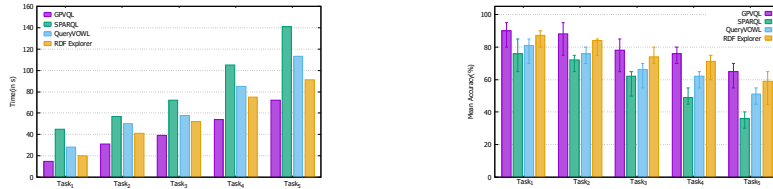
	Task 1	Task 2	Task 3
Description	Find philosophers influenced by Bob Black.	Find people influenced Bob Black and Karl Marx.	Find people who influenced both Bob Black and people influenced by Bob Black.
Query pattern			
	Task 4	Task 5	
Description	Find year of birth and death of people who influenced both Bob Black and people influenced by Bob Black.	Find people whose birth date is after 1750 and influenced philosophers that influenced both Bob Black and people influenced by Bob Black.	
Query pattern			

We ranked the difficulty of each task based on the number of nodes, edges, and constraints needed. As can be seen in Table 2, the tasks are related to each other, and that later task can be built incrementally from previous tasks. This design is intended to simulate the actual workflow in a real-world setting, where an end-user typically writes related and incrementally more complex queries, but not unrelated random ones.

From Fig. 3, we can obtain the following conclusions: (1) Among the five tasks, the query completion times of GPVQL are the shortest; (2) As the tasks become more difficult, the query completion times (resp. accuracy) of SPARQL are gradually increasing (resp. reducing). However, GPVQL outperforms all the other systems: SPARQL, QueryVOWL, and RDF Explorer, and the accuracy of GPVQL remains above 60%. As shown in Fig. 3(b), GPVQL moderately outperforms QueryVOWL on accuracy for all tasks, and significantly outperforms QueryVOWL for the more complex ones, i.e., Task 4 and Task 5. Compared

³ <http://gpvql.gq/>

⁴ <http://dbpedia.org/sparql/>



(a) Mean completion time for each tool per task. (b) Mean accuracy for each tool per task.

Fig. 3. Experiment results on completion time and accuracy.

with RDF Explorer, GPVQL reduces the completion time by approx. 15%, and increases the accuracy by about 10%. In summary, our evaluation demonstrates the superiority of GPVQL in both aspects of completion time and accuracy. We believe that the main reasons of GPVQL’s superiority are bidirectional transformation and the deep exploration it facilitates. For changes of the query tasks, traditional query languages requires users to construct new queries from scratch, which reduces the speed, usability, and user-friendliness. With the bidirectional transformation in GPVQL, users can use query results of the previous query pattern as the input of the next query pattern, which reduces the difficulty and time used of creating queries.

4 Conclusion

In this paper, we propose a general-purpose visual query language. The main advantage of GPVQL is the flexible bidirectional transformation between query patterns and graph results, which is a useful method for end-users to gain insights of large-scale knowledge graphs, and eliminates the boundary between query patterns and graph results. We experimentally validated the accurateness and effectiveness of GPVQL and its associated interface, showing its superiority over other visual query languages.

References

1. Haag, F., Lohmann, S., Siek, S., Ertl, T.: Queryvowl: A visual query notation for linked data. In: International Semantic Web Conference. pp. 387–402. Springer (2015)
2. Jayaram, N., Khan, A., Li, C., Yan, X., Elmasri, R.: Querying knowledge graphs by example entity tuples. *IEEE Transactions on Knowledge and Data Engineering* **27**(10), 2797–2811 (2015)
3. Vargas, H., Buil-Aranda, C., Hogan, A., López, C.: Rdf explorer: A visual sparql query builder. In: International Semantic Web Conference. pp. 647–663. Springer (2019)