KG3D: An Interactive 3D Visualization Tool for Knowledge Graphs

Dawei Xu[1,2], Lin Wang[1], Xin Wang[2](✉), Dianquan Li[1], Jianpeng Duan[1], and Yongzhe Jia[1]

[1] TechFantasy Co., Ltd., Tianjin 300387, China
{xudawei,linwang,lidianquan,duanjianpeng,jia}@techfantasy.cn
[2] College of Intelligence and Computing, Tianjin University, Tianjin 300072, China
wangx@tju.edu.cn

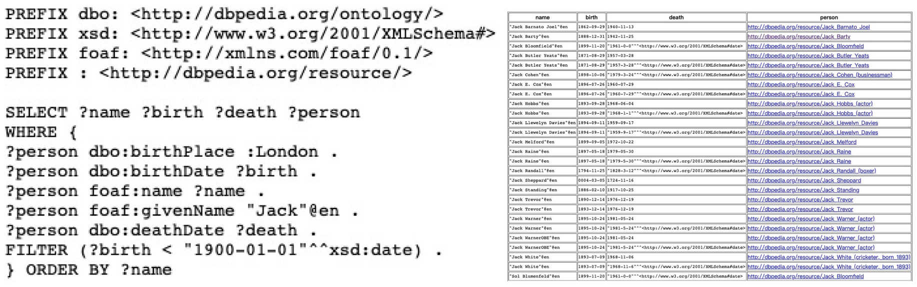
Abstract. With the emerge of knowledge graphs in different scales like DBpedia, YAGO, and WikiData, they have become the cornerstone to support many artificial intelligence tasks. However, it is difficult for end-users to query and understand those knowledge graphs consisting of hundreds of millions of nodes and edges. To help end-users better retrieve information from RDF data and explore the knowledge graph without SPARQL or knowing the relation types, we developed an interactive visual query tool, called KG3D, which can realize connection query and pattern matching. Our tool can view the knowledge graph in 3-dimensional space and automatically convert the query to the SPARQL statement. In this paper, we present the superiority of KG3D over other tools, discuss the design motivation, and demonstrate various use cases.

Keywords: Knowledge graph · Semantic web · Visualization · RDF · SPARQL · WebGL

1 Introduction

The Resource Description Framework (RDF) is a standard model for describing resources and exchanging data on the Web [1], which has a feature that facilitates data integration in different schemas. These features make RDF widely developed and accumulated a large number of knowledge graphs like DBpedia, YAGO, and WikiData [2]. SPARQL has emerged as the standard RDF query language by the World Wide Web Consortium. SPARQL is a SQL-like query language able to retrieve and manipulate data stored in RDF. With this language, users can pull values from structured and semi-structured data and explore data by querying unknown relations [3]. However, it is unrealistic to ask end-users to learn SPARQL to query the information they need, which reduces the availability of knowledge graphs for most users. For example, we want to find people born in London with the given name "Jack" before 1900. To issue this query, we have a SPARQL and the corresponding results as shown in Fig. 1.

J. Li et al. (Eds.): ADMA 2019, LNAI 11888, pp. 886–889, 2019.
https://doi.org/10.1007/978-3-030-35231-8_67



```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://dbpedia.org/resource/>

SELECT ?name ?birth ?death ?person
WHERE {
?person dbo:birthPlace :London .
?person dbo:birthDate ?birth .
?person foaf:name ?name .
?person foaf:givenName "Jack"@en .
?person dbo:deathDate ?death .
FILTER (?birth < "1900-01-01"^^xsd:date) .
} ORDER BY ?name
```

name	birth	death	person
"Jack Barnato Joel"@en	1862-09-29	1940-11-13	http://dbpedia.org/resource/Jack_Barnato_Joel
"Jack Barty"@en	1888-12-31	1942-11-25	http://dbpedia.org/resource/Jack_Barty
"Jack Bloomfield"@en	1899-11-20	"1961-0-0"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Bloomfield
"Jack Butler Yeats"@en	1871-08-29	1957-03-28	http://dbpedia.org/resource/Jack_Butler_Yeats
"Jack Butler Yeats"@en	1871-08-29	"1957-3-28"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Butler_Yeats
"Jack Cohen"@en	1898-10-06	"1979-3-24"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Cohen_(businessman)
"Jack E. Cox"@en	1896-07-26	1960-07-29	http://dbpedia.org/resource/Jack_E._Cox
"Jack E. Cox"@en	1896-07-26	"1960-7-29"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_E._Cox
"Jack Hobbs"@en	1893-09-28	1968-06-04	http://dbpedia.org/resource/Jack_Hobbs_(actor)
"Jack Hobbs"@en	1893-09-28	"1968-1-1"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Hobbs_(actor)
"Jack Llewelyn Davies"@en	1894-09-11	1959-09-17	http://dbpedia.org/resource/Jack_Llewelyn_Davies
"Jack Llewelyn Davies"@en	1894-09-11	"1959-9-17"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Llewelyn_Davies
"Jack Melford"@en	1899-09-05	1972-10-22	http://dbpedia.org/resource/Jack_Melford
"Jack Raine"@en	1897-05-18	1979-05-30	http://dbpedia.org/resource/Jack_Raine
"Jack Raine"@en	1897-05-18	"1979-5-30"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Raine
"Jack Randall"@en	1794-11-25	"1828-3-12"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Randall_(boxer)
"Jack Sheppard"@en	0004-03-05	1724-11-16	http://dbpedia.org/resource/Jack_Sheppard
"Jack Standing"@en	1886-02-10	1917-10-25	http://dbpedia.org/resource/Jack_Standing
"Jack Trevor"@en	1890-12-14	1976-12-19	http://dbpedia.org/resource/Jack_Trevor
"Jack Trevor"@en	1893-12-14	1976-12-19	http://dbpedia.org/resource/Jack_Trevor
"Jack Warner"@en	1895-10-24	1981-05-24	http://dbpedia.org/resource/Jack_Warner_(actor)
"Jack Warner"@en	1895-10-24	"1981-5-24"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Warner_(actor)
"Jack WarnerOBE"@en	1895-10-24	1981-05-24	http://dbpedia.org/resource/Jack_Warner_(actor)
"Jack WarnerOBE"@en	1895-10-24	"1981-5-24"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Warner_(actor)
"Jack White"@en	1893-07-09	1968-11-06	http://dbpedia.org/resource/Jack_White_(cricketer,_born_1893)
"Jack White"@en	1893-07-09	"1968-11-6"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_White_(cricketer,_born_1893)
"Sol Blumenfeld"@en	1899-11-20	"1961-0-0"^^<http://www.w3.org/2001/XMLSchema#date>	http://dbpedia.org/resource/Jack_Bloomfield

Fig. 1. SPARQL statements and the corresponding results.

In the example, we can find the following problems: First, the SPARQL query is difficult for end users to learn to express their query demands. Second, SPARQL WHERE clauses, abbreviations with prefixes, and full URIs are verbose in the statements. Third, users are supposed to know what relations a specific entity has before the query. The tool should prompt a list of possible relations before the input. Last, most of the visualization tools present results in 2-dimension. Due to space constraints, entities and relations in large number have a high probability of overlapping, which may affect visualization effects. Thus, there is a massive gap between end users and understanding RDF data.

Our research focuses on the design and implementation of a new interactive method of visualization and query for the RDF knowledge graphs. Named KG3D, our demo prototype overcomes the deficiencies above.[1] Our tools can view the knowledge graph in 3-dimensional space by changing the angle of view from overview to details. So the vast spatial and physical force settings ensure that overlap does not occur. The query can automatically convert to SPARQL and full URI to the endpoints, with a heuristic relation prompt function.

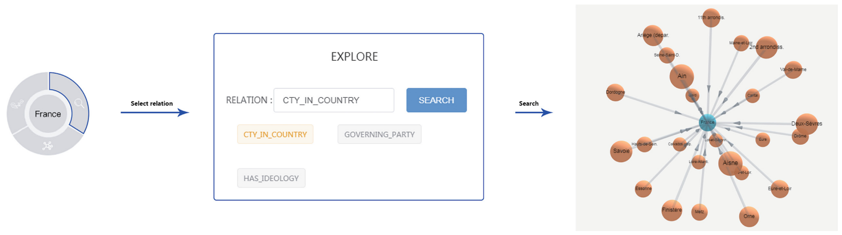
2 Demonstration

The data shown in this demo are collected from the SPARQL endpoint of DBpedia. We demonstrate the tool by the following use cases based on the European politics dataset, which contains European cities, the political parties in their local governments, and their ideologies.

Use Case 1: Entity Visual Query. This tool can be used as DBpedia that users are allowed to search for an entity and get its neighbors. When it comes to multiple types of relations, a query can become complicated. This tool as well provides a function that queries a node's neighboring entities by its relation types. For example, "How many cities in France?" To find an answer, we first need to find France, then find those cities in connection with it. We can input

[1] GitHub repository: https://github.com/selenesoft/kg3d.

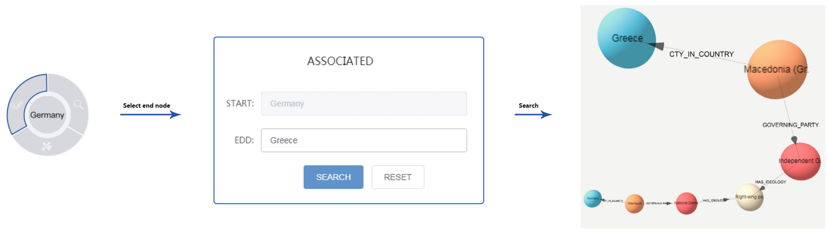
the name in the search box and click the search button. The entity returns as the query result. To find neighbor nodes, user can click on the entity's circular menu and select the explore button, and a pop-up window shows all possible relation types. Select the types by the checkboxes, then execute the query. As shown in Fig. 2, linked nodes and directed edges are shown as user selected. We can repeat this operation to all nodes on the screen.



RELATION : CTY_IN_COUNTRY

Fig. 2. Querying entity and its neighbours.

Use Case 2: Connection Query. An important aspect of exploratory search over graph data is to understand what paths connect a given pair of nodes. For example, we want to know how Greece and Germany have parties that share the same ideology. This question can be converted to a problem that determines whether there exists a path between two countries via parties and ideologies. A user starts from a click on one entity's circular menu and selects the path query button on the upper left. We should fill the detail conditions in the text box. The node clicked is automatically set as the object of the first clause. Another entity's name can be input as the second clause, which is the terminal of the path. The query results are shown as Fig. 3.



START: Germany
EDD: Greece

Fig. 3. Finding a path from France to Greece.

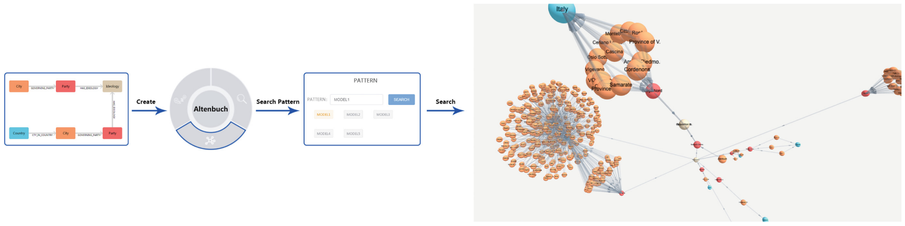


Fig. 4. The process of the complete pattern query.

Use Case 3: Pattern Exploration. If there exists only one type of "Father-Son" relation, we can find someone's grandfather by his father's father. That is a set of conditional clauses or a pattern. This function provides a method to help users complete a set of conditional clauses to query all qualified entities and relations. For example, if we want to know which country share the same ideology with the city Altenbuch's governing party, then the query pattern should be the party that runs the city, party's ideologies, other parties with the same ideology, those parties' governing cities, and corresponding countries. Thus, open the pattern design box. First, drag and drop a city node from the left panel of the screen as the object of the first clause. Then we can click the node to add a relation, which should point to a party node. These operations should be repeated to add more condition clauses until we finish building the pattern, as shown in Fig. 4. We can apply the pattern to the city node named Altenbuch to get the query results. Users can also change the view of the results from a graph to the table if they do not want to see a complex graph with too many nodes.

3 Conclusion

We proposed an interactive visualization tool for presenting RDF data in 3-dimensional space. This tool can help users better retrieve information from RDF data and explore the knowledge graphs without being trained to use SPARQL or knowing the relation types. To enhance usability, we also introduced advanced functions including connection query and pattern matching.

References

1. Wang, X., Wang, J.: ProvRPQ: an interactive tool for provenance-aware regular path queries on RDF graphs. In: Cheema, M.A., Zhang, W., Chang, L. (eds.) ADC 2016. LNCS, vol. 9877, pp. 480–484. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46922-5_44
2. Yang, C., Wang, X., Xu, Q., Li, W.: SPARQLVis: an interactive visualization tool for knowledge graphs. In: Cai, Y., Ishikawa, Y., Xu, J. (eds.) APWeb-WAIM 2018. LNCS, vol. 10987, pp. 471–474. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96890-2_41
3. Gómez-Romero, J., et al.: Visualizing large knowledge graphs: a performance analysis. Future Gener. Comput. Syst. **89**, 224–238 (2018)