Fuzzy Systems and Data Mining IX
A.J. Tallón-Ballesteros and R. Beltrán-Barba (Eds.)

doi:10.3233/FAIA231019

Scalable Interactive Keyword Query Interface over Knowledge Graph

Xin Hu, Jiangli Duan*, Sulan Zhang
Yangtze Normal University, Chongqing, China

Abstract. Abstract goes here. Existing keyword query systems over knowledge graph can produce interesting results and are easy to use. However, they cannot handle the ambiguities that have matches in the knowledge graph, namely, multiple interpretations may be correct so that they cannot determine which interpretation is what the user expects. And they cannot scale to handle the knowledge graphs with more than billions of triples or thousands of types/predicates. On the one hand, we construct an interactive interface in which the above ambiguities will resort to the user. To enhance the user experience, we formalize the interaction problem and then propose an algorithm to find a best scheme of interaction (i.e., a verifying sequence with lowest interaction times and candidates) based on the dependency relations between mappings. On the other hand, we propose a new schema graph, i.e., type-predicate graph, which has good scalability while containing complete information for building query graph. No matter how large the knowledge graph is, the size of type-predicate graph is always very small because its size depends on the number of types and predicates whose number are far less than that of triples in knowledge graph. Finally, we have demonstrated our contributions with several well-directed experiments over real datasets (DBpedia and Yago).

Keywords. Artificial intelligence; knowledge graph; keyword query; interactive interface; type-predicate graph

1. Introduction

Keyword query is a useful tool for exploring large knowledge graphs, as it is user-friendly. Without requiring users to master the domain knowledge of the data schema and syntax of SPARQL, the keyword query technique can easily return satisfactory query results by only specifying a few enquiry keywords, e.g., “China capital”. Existing keyword query systems over knowledge graph can produce interesting results, but they suffer from limitations as follows.

Existing keyword query systems [1-12] enable users to query information in knowledge graph by returning the subgraph containing the keywords, but they may return unwanted answers because there are too many possible interpretations. For query “Feng_xiaogang films”, it is not easy to answer this query since there are too many paths between the entity “Feng_xiaogang” and the instances of the type “Film” (e.g., the paths “-direct/starringIn-,”“-direct-,” “-starringIn-,” “-award-,” “-FilmDirector,” “-spouse-x-starringIn-” as shown in Figure 6). Existing techniques can filter out some false and valueless interpretations, but for valuable interpretations matching knowledge graph,

* Corresponding author. Jiangli Duan, Yangtze Normal University, Chongqing, China; E-mail addresses: duanjl@yznu.edu.cn

they could do no more. For instance, the entity recognition technique can recognize the entity "Feng_xiaogang", the phrase mapping technique can map "films" to the type "Film", and the backward search technique can filter out the interpretation "Feng_xiaogang-FilmDirector" and "Feng_xiaogang-spouse-x-starringIn-film" by score and so on. However, the 4 interpretations in Figure 1 cannot be filtered out by existing techniques because each interpretation has at least one matching subgraph in the knowledge graph, that is, any one may be users' expectation.

Keyword Query: *Feng_xiaogang films*
Interpretation 1:
All films that are directed and acted by Feng_xiaogang.
Interpretation 2:
All films that are directed by Feng_xiaogang.
Interpretation 3:
All films that are acted by Feng_xiaogang.
Interpretation 4:
All filmaward which Feng_xiaogang was awarded.

Figure 1. Possible interpretations

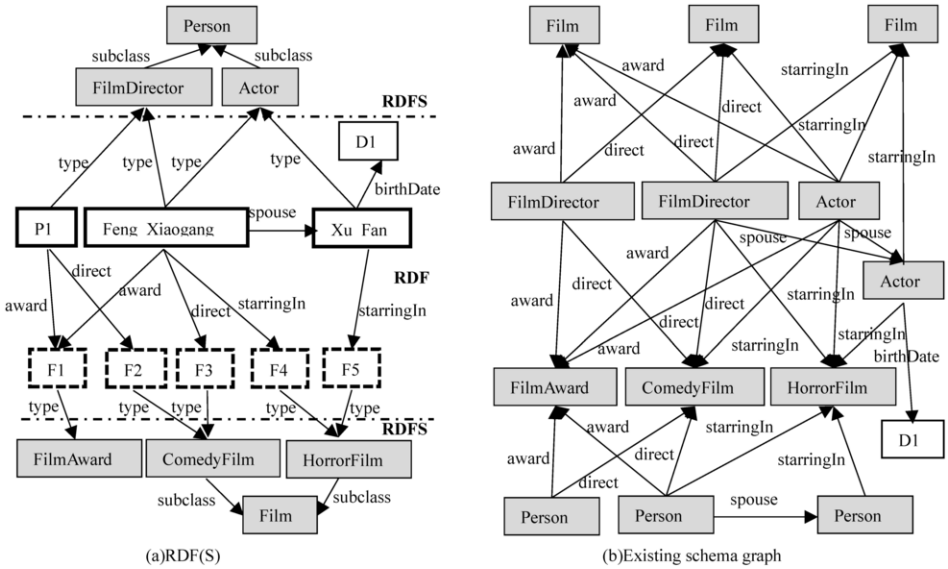

Figure 2. RDF(S) and existing schema graph

Existing keyword query systems cannot scale to handle the knowledge graphs with more than billions of triples or thousands of types/predicates. They can be divided into two categories: **1)** ***data index*.** The prevalent approaches [1-8] building on dedicated indexing techniques aim at finding substructures that connect the data elements which match the keywords. With the explosive growth of knowledge graph, it is obvious that the dedicated data index will be faced with bottleneck, especially knowledge graph with billions of triples. **2)** ***schema graph*.** Some systems [9-12] build schema graph to capture the "schema" information in knowledge graph and then automatically generate query graphs (which is used to produce SPARQL statements) by schema graph. However, these systems capture all relations without standardizing them, so that each type/predicate will appear many times in schema graph as **Figure 2**(b), which will lead to that the volume of

schema graph will increase exponentially. Moreover, the schema-free nature of RDFS enables knowledge graph has great power of expression, while it also makes RDFS more complex, as shown in **Figure 2**(a): 1) The types have multiple levels in RDFS; 2) One entity may have multiple types; 3) The RDFS contains a large amount of data.

In our daily communication, if we are not sure of the meaning of a question, we will make the question clear by asking some related questions back to the asker. Inspired by the daily communication habits [13,14], we propose an interactive keyword query interface, which filters out the query graphs (i.e., query interpretations) that has matches in knowledge graph but don't correspond to expectation of the user. Different from existing interactive query methods [15-17] that has a bad user experience because they build query almost entirely depended on user, in our framework, most of the process of building query is resorted to computer, and the ambiguousness that existing techniques cannot filter out are resorted to user, which can improve user experience. Moreover, we propose a scalable schema graph (i.e., a type-predicate graph) to support the process of producing query graph, where query graph is the bridge between keyword query and SPARQL statement. We make the following contributions in this paper:

1) In the offline phase, we propose a scalable schema graph (i.e., a type-predicate graph) over knowledge graph, which contains complete information for building query graph and has good scalability because its size depends on the types and predicates.

2) To handle the ambiguities that have matches in knowledge graph, we construct an interactive interface, in which part of the process of building query be resorted to computer, and the ambiguities that existing method cannot handle will resort to the user by presenting user with the ambiguous candidates and letting user make choice.

3) To enhancing the user experience during the verification of the ambiguities, we formalize the interaction problem and propose an algorithm to find a best scheme of interaction (i.e., a verifying sequence with lowest interaction times and candidates) based on dependency relations between mappings.

2. Type-Predicate Graph

As mentioned in the motivating example, existing methods concerning schema graph cannot scale to handle the knowledge graph with more than thousands of types or predicates. In contrast, we construct a type-predicate graph consisting of relationships between types and predicates, which has a smaller amount of data and contains complete information for building query graph. With the type-predicate graph, adjacent predicates and types for any type or predicate can be efficiently retrieved from small amounts of data rather than knowledge graph with billions of triples.

2.1 Extracting Relationship Subgraph

Relationship subgraph consists of types and predicates from the triples. For instance, as shown in Figure 2(a), the entity "Feng_Xiaogang" has three types, i.e., "Person," "FilmDirector" and "Actor". Except for "type", "Feng_Xiaogang" has four predicates (i.e., "spouse," "direct," "starringIn" and "award") corresponding to four entities (i.e., "F1," "F3," "F4" and "Xu_Fan"), and these entities also have their own type set, i.e., "Person,Actor," "Film,ComedyFilm," "Film, HorrorFilm" and "FilmAward". From these, we can obtain the first adjacent relationships subgraph between types and

predicates as shown in Figure 3. In the same way, for entities P1/Xu_Fan, two other relationship subgraphs in Figure 3 can be obtained.

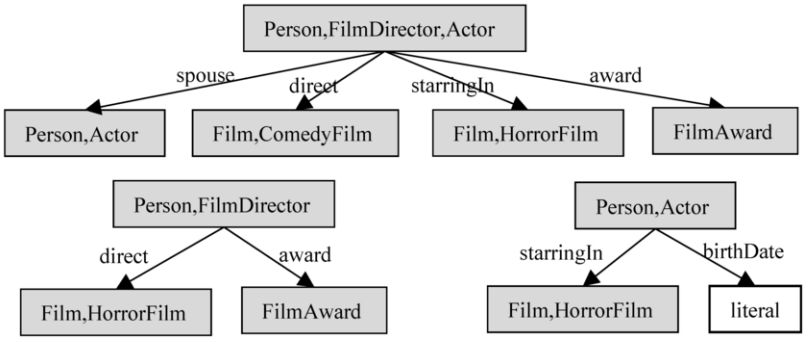

Figure 3. Extracted Relationship Subgraph

2.2 Standardization

The number of types is far less than that of triples in knowledge graph, so the amount of data in the relationship subgraphs will be small if any type only has one corresponding relationship subgraph. For the relationship subgraphs with same type, by combining all outgoing adjacent predicates and counting the number of repeat times for each outgoing edge, we can obtain one standardized relationship subgraph, e.g., Figure 4.

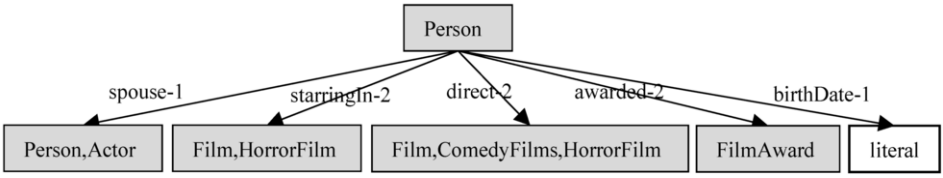

Figure 4. Standardized Relationship Subgraph

2.3 Type-Predicate Graph

All standardized relationship subgraphs form the type-predicate graph, where these subgraphs link to each other by the same types, but we do not actually connect them, as shown in Figure 5. From the type-predicate graph, for any type or predicate, all its adjacent types or predicates can be found easily.

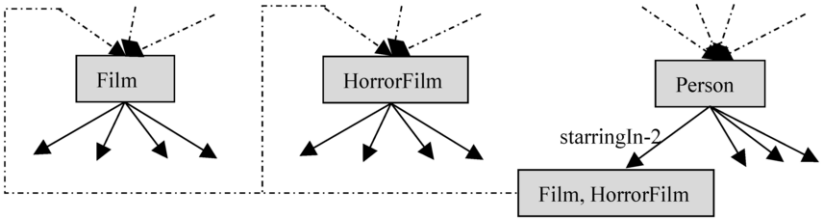

Figure 5. Type-Predicate Graph

3. Interactive Keyword Query Interface

As mentioned in motivating example, although existing methods concerning keyword query can filter out false interpretations, they cannot handle the ambiguities that have matches in knowledge graph. In contrast, we will construct an interactive interface in which the ambiguities will resort to the user by presenting the ambiguous candidates and letting user make choice. Furthermore, to enhance the user experience, we formalize the interaction problem and then propose an algorithm to find a best scheme of interaction. In this section, we first introduce the general process of keyword query in existing methods and then propose the scheme of interaction.

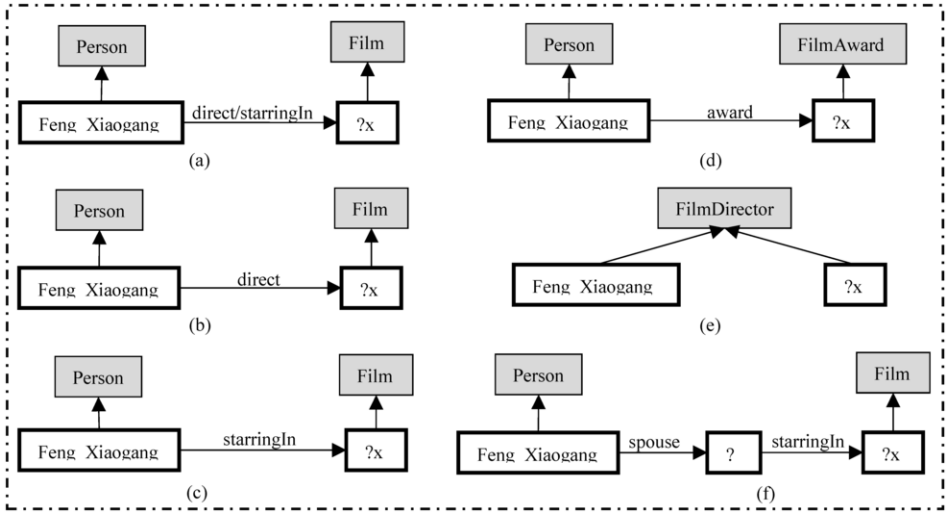

Figure 6. Query graphs for sample query

3.1 The General Process of Keyword Query

Existing methods first find all candidate mappings for each keyword in keyword query, then construct query graphs by schema graph and finally translate query graphs into SPARQL statements [18].

Example 1. For sample query "Feng_xiaogang films", firstly, existing methods obtain candidate entities (e.g., "Feng_Xiaogang" and "Feng_xiaogangX") and their corresponding types (e.g., "FilmDirector," "Actor" and "Person" for the former, and "Book" for the latter) for the keyword "Feng_xiaogang", and candidate types (e.g., "FilmDirector," "FilmAward," "ComedyFilm," "HorrorFilm" and "Film") for the keyword "films". Secondly, they combine candidate mappings of all keywords to obtain a set of query graphs by schema graph. During combination, the ambiguities without matches in knowledge graph and some query graphs with low value are deleted, e.g., Figure 6(e) and Figure 6(f). Thirdly, they translate remained query graphs, e.g., Figure 6(a)/(b)/(c)/(d), to SPARQL statements and then return answers.

3.2 Scheme of Interaction

Existing methods can delete valueless candidates, i.e., ambiguities that has no matches in knowledge graph and query graph with low value (Example 1). However, if multiple

query graphs match the subgraphs in the knowledge graph, they cannot recognize which one is user's expectation. This paper will resort to user by presenting the ambiguous candidates and letting user make choice. To enhance the user experience, we formalize the interaction problem and propose an algorithm to find a best scheme of interaction.

Definition 1. (Interaction Diagram) An interaction diagram is a tuple G=(V, E, N). 1) Each vertex $v \in V$ represents an object that needs to be verified. 2) E includes directed edges and undirected edges. Directed edge $<v_1, v_2> \in E$ means we don't need to verify v_2 if v_1 is verified to be true. Undirected edge $<v_1, v_2> \in E$ represents that v_1 and v_2 are adjacent in query graph, which will be omitted if there is a directed edge. 3) $c(v) \in N$ represents the number of candidate mappings of vertex v.

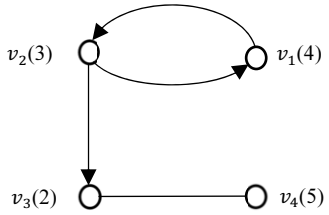

Figure 7. An interaction diagram

Example 2. Figure 7 presents an example of interaction diagram, where $v_3(2)$ represents that vertex v_3 has 2 candidate mappings (e.g., keyword "films" has two mappings "Film" and "FilmAward" in retained valuable query graphs in Figure 6, where Figure 6(e) and Figure 6(f) were deleted in Example 1).

Since each vertex $v \in V$ represents an object that needs the user to verify which candidate mapping is correct. For a set of vertices V= $\{v_1, v_2, \ldots, v_n\}$, in general, there are n! possible verifying sequences.

Definition 2. (Possible/Verifying Sequence, *ps*/*vs*). *ps* is a set of all vertices in an interaction diagram G, and these vertices have a deterministic order. All vertices that need to be verified form a verifying sequence *vs*.

Definition 3. (Calls/Candidates of a verifying sequence, *Calls*/*Cans*). Given a verifying sequence *vs*, we count the interaction times, denoted as *Calls*(*vs*), and the number of candidates of necessary interaction, denoted as *Cans*(*vs*).

Example 3. Table 1 presents all verifying sequences for the interaction diagram in Figure 7. From Figure 7, verifying v_4 doesn't affect other vertex (i.e., undirected edge (v_3, v_4)) so that we only consider the order for other three vertices $v_1/v_2/v_3$ in Table 1. For the possible sequence *ps*= $\{v_1, v_2, v_3, v_4\}$, there are two directed edges (i.e., $<v_1, v_2>$ and $<v_2, v_3>$) so that both v_2 and v_3 don't need to be verified after verifying v_1. Thus, we obtain the verifying sequence *vs*= $\{v_1, v_4\}$, and then *Calls*(*s*)=2 and *Cans*(*s*)=9 (i.e., v_1 and v_4 have 4 and 5 candidate mappings, respectively). Moreover, to find a best verifying sequence, we first find the verifying sequences that have lowest *Calls* (i.e., *Calls*(*vs*)=2), and then we select one of them with lowest *Cans* (i.e., *Cans*(*vs*)=8), so verifying sequence with lowest *Calls*/*Cans* is *vs*= $\{v_2, v_4\}$.

Table 1. A set of verifying sequences

ps	*vs*	*Calls*	*Cans*
$v_1, v_2, v_3, (v_4)$	v_1, v_4	2	9
$v_1, v_3, v_2, (v_4)$	v_1, v_4	2	9
$v_2, v_1, v_3, (v_4)$	v_2, v_4	2	8
$v_2, v_3, v_1, (v_4)$	v_2, v_4	2	8
$v_3, v_1, v_2, (v_4)$	v_3, v_1, v_4	3	11
$v_3, v_2, v_1, (v_4)$	v_3, v_2, v_4	3	12

Definition 4. (Interaction Problem). Given an interaction diagram, interaction problem is to find a verifying sequence with lowest *Calls*/*Cans*.

During solving interaction problem, several issues are worthy of considering: 1) different from topological graph, in interaction diagram or its subgraph, the loop may exist, and the vertex whose indegree is zero may not exist; 2) to obtain a verifying sequence, we should delete the vertices that doesn't need to be verified from the possible sequences; 3) it is necessary to obtain all verifying sequences with lowest *Calls*, because *Cans* needs to be contrasted when several verifying sequences has same lowest *Calls*.

Therefore, we divide the process of solving interaction problem into two stages that correspond to two sub-algorithms, i.e., Algorithm FPS and Algorithm FVS. Algorithm FPS outlines the process of finding all possible sequences from interaction diagram. Based on recursive function *FPS*(), if there is a vertex whose indegree is zero, adding it to *ps* should be a top priority. Otherwise, we select suboptimal scheme, namely, adding the vertex whose outdegree is not zero to *ps*, which lead to that we can delete one vertex at least in the process from *ps* to *vs*. Moreover, if there is no directed edge in *G*, we add all vertices to *ps* and stop the recursion of this branch.

Algorithm FPS (Finding all possible sequences)

Input: *G*: the interaction diagram
Output: *PS*: the set of all possible sequences
Values: *ps*: a possible sequence; *n*: the number of vertices in *G*; v_i: the *i*-th vertex in *G*.

```
1:  ps={};n=Get_count_vertices(G);
2:  FPS(n, G, ps);
3:    If (n>0)
4:      If (there is no directed edge in G)
5:        For (i=0; i<n; i++)
6:          ps=ps+v_i;
7:        Adding ps to PS;
8:      Else For (flag=0, i=0; i<n; i++)
9:          If (indegree of v_i is zero) FPS(n-1, G-v_i, ps+v_i); flag=1;
10:       If (flag == 0)
11:         For (i=0; i<n; i++)
12:           If (outdegree of v_i is not zero) FPS(n-1, G-v_i, ps+v_i);
13:   Else Adding ps to PS;
```

Algorithm FVS outlines the process of finding the verifying sequences with lowest *Calls*/*Cans*: 1) obtaining the verifying sequences by deleting the vertices that doesn't need to be verified from possible sequences; 2) retaining the verifying sequences with lowest *Calls*; 3) for the verifying sequences with same lowest *Calls*, calculating their *Cans* and then retaining the verifying sequence with lowest *Cans*.

Algorithm FVS (Finding Verifying Sequence with lowest Calls)

Input: *E*: the set of directed edges
PS: the set of all possible sequences
Output: *BVS:* the set of verifying sequences with lowest *Calls/Cans*
Values: *count_min*: currently minimum; *VS*: a set of verifying sequences; *count_ps*: the number of possible sequences; *ps*: the *i*-th possible sequence in *PS*; *count_v*: the number of vertices in *ps*; v_i,v_j: a vertex in *ps*.

```
1:  count_min=∞; VS={}
2:  count_ps=Get_count_ps(PS)
3:  For (i=0; i<count_ps; i++)
4:      ps=PS[i]; count_v=Get_count_vertices(ps);
5:      For (i=count_v-1; i>0; i--)
6:          For (j=count_v-2; j>0; j--)
7:              If(directed edge <v_j, v_i>∈E) Delete v_i from ps; count_v--; Break;
8:      If(count_v<count_min) count_min=count_v; Deleting all ps in VS; Adding ps to VS;
9:      Else if (Count_v==Count_min) Adding ps to VS;
10: Calculating Cans of all ps in VS
11: Selecting the ps with lowest Cans as BVS
```

4. Experimental Evaluation

We have demonstrated our contributions with several well-directed experiments over real datasets (DBpedia and Yago). First, we propose the type-predicate graph that has good scalability and was used to build query graph, so we explain the scalability and the filtering capability of the type-predicate graph. Second, we construct an interactive interface to handle the ambiguities that have matches in the knowledge graph, so we contrast keyword query with or without interaction to demonstrate the interaction capability. Third, we propose an algorithm to find a best verifying sequence, so we show the optimization capability of the algorithm.

4.1 The Scalability of The Type-Predicate Graph

Type-predicate graph has good scalability because its size depends on the number of types and predicates. Table 2 shows the sizes of Yago (core) dataset, DBpedia (infobox) dataset and the type-predicate graph. From triples ratio, the number of triples in type-predicate graph is far less than that in Yago and DBpedia. Moreover, the number of relationship subgraphs is equal to the number of types because the relationship subgraph is dominated by the types, and the size of graph triples depends on the number of types and predicates because graph triples are the combinations of types and predicates. Since the number of types and predicates is far less than the number of triples in a knowledge graph, the size of type-predicate graph is far smaller than that of knowledge graph.

Table 2. Data size

		YAGO (core)	DBpedia (infobox)
knowledge graph	Data triples	45453166	64813068
	Types	347868	418
	Predicates	70	46510
Type-predicate graph	Relationship Subgraphs	347868	418
	Graph triples	1978891	126637
	Triples Ratio	4.35%	0.19%

4.2 The Filtering Capability of The Type-Predicate Graph

During building query graph, type-predicate graph can filter out some inappropriate candidate mappings as shown in Table 3. For instance, for the mappings of keyword "produce" in query "feng_xiaogang, produce, film", there are 182 predicates containing the string "produce" in DBpedia (infobox) dataset. Among of them, in the relationship subgraph whose main type is "Film", there are 73 predicates containing the string "produce" so that we can delete 109 (182-73) predicates. Furthermore, between type "Film" and "Person", there are only 24 predicates containing string "produce". And then, we use existing techniques (e.g., similarity scores, interaction and so on) to select candidate mappings. In conclusion, type-predicate graph has filtering capability so that we can obtain more suitable candidate predicates than existing methods.

Table 3. The filtering capability of type-predicate graph

triples whose predicate contains string "*produce*" in DBpedia	number
?*subject*, ****produce***, ?*object*	182
?*subject*(whose type is "*Film*"), ****produce***, ?*object*	73
?*subject*(whose type is "*Film*"), ****produce***, ?object(whose type is "*Person*")	24

Table 4. The sample queries

Keyword queries	Query intention
Feng Xiaogang, film	*All film whose actor (or director) is Feng xiaogang.*
Free University, Amsterdam, students	*The number of students in the Free University, Amsterdam.*
car, Germany	*All cars that are produced in Germany.*
people, born, Vienna, die, Berlin	*All people that were born in Vienna and died in Berlin.*
actor, movie, direct, starring, William Shatner	*All actors starring in movies directed by and starring William Shatner.*

4.3 The Interaction Capability

With interaction, users can fully express their expectations. Existing keyword query methods (i.e., keyword query method without interaction) always return top-k results for one keyword query, and if the top-k set contains the correct one, it is considered to be able to answer this keyword query. In contrast, during the process from query to result, our method enable user to select query graph that satisfies their expectations by interaction, so the correct one can be always selected. As shown in Figure 8, we show the rank number of correct query graph in top-k set by existing methods (i.e., without interaction) and our method (i.e., with interaction), where the keyword queries come from Table 4.

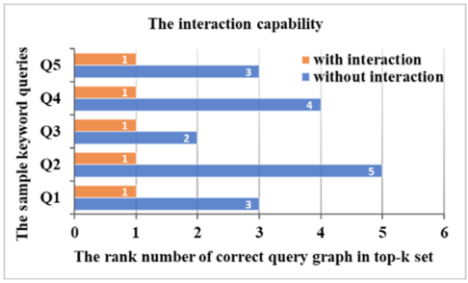

Figure 8. The interaction capability

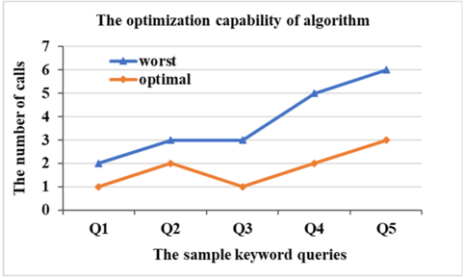

Figure 9. The optimization capability of algorithm

4.4 The Optimization Capability of Algorithm

To enhance the user experience, we formalize the interaction problem and then propose an algorithm to find a verifying sequence with lowest *Calls*/*Cans*. For an interaction diagram, there are multiple possible verifying sequences. We show the number of *calls* (i.e., interaction times) in worst/optimal case as shown in Figure 9, and the algorithm always can obtain the optimal verifying sequence for the keyword queries come from Table 4. Moreover, the increasing number of keywords in queries leads to increase number of possible candidate combinations, so interaction times (i.e., *calls*) will increase, but it is not absolute (e.g., $Q2$ and $Q4$ have 3 and 5 keywords, respectively. However, they have same number of *calls*.).

5. Conclusions

Although existing keyword query systems over knowledge graph can produce interesting results and are easy to use, they cannot handle the ambiguities that have matches in knowledge graph and cannot scale to handle the knowledge graph with more than billions of triples or thousands of types/predicates. So, we propose an interactive keyword query

interface with type-predicate graph, which handle above ambiguities by a best scheme of interaction, and type-predicate graph enables keyword query can scale to handle various huge knowledge graphs. At last, we have demonstrated our contributions with several well-directed experiments over real datasets (DBpedia and Yago).

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62106024, No. 62006029), the China Postdoctoral Science Foundation (No. 2022M711458), the Natural Science Foundation of Chongqing (No. CSTB2022NSCQ-BHX0018), the Chongqing Language Research Project (No. yyk22105), and the youth project of Chongqing Education Commission of China (No. KJQN202201410, No. KJQN202201413, KJQN202001434).

References

[1] Cheng G, Li S, Zhang K, Li C. Generating compact and relaxable answers to keyword queries over knowledge graphs. Proceedings of the 19th International Semantic Web Conference; 2020. p.110-127.
[2] Hu X, Duan J, Dang D. Natural language question answering over knowledge graph: The marriage of SPARQL query and keyword search. Knowledge and Information Systems. 2021; 63(4): 819-844.
[3] Nikas C, Fafalios P, Tzitzikas Y. Open domain question answering over knowledge graphs using keyword search, answer type prediction, sparql and pre-trained neural models. Proceedings of the International Semantic Web Conference; 2021. p.235-251.
[4] Shan Y, Li M, Chen Y, Constructing target-aware results for keyword search on knowledge graphs. Data & Knowledge Engineering. 2017; 110:1-23.
[5] Shi Y, Cheng G, Kharlamov E. Keyword search over knowledge graphs via static and dynamic hub labelings. Proceedings of the International World Wide Web Conference; 2020. p.235-245.
[6] Shi Y, Cheng G, Tran TK, Tang J, Kharlamov E. Keyword-based knowledge graph exploration based on quadratic group steiner trees. Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence; 2021. p.1555-1562.
[7] Shi Y, Cheng G, Tran TK, Tang J, Kharlamov E, Shen Y. Efficient computation of semantically cohesive subgraphs for keyword-based knowledge graph exploration. Proceedings of the International World Wide Web Conference; 2021. p.1410-1421.
[8] Yang Y, Agrawal D, Jagadish HV, Tung KH, Wu S. An efficient parallel keyword search engine on knowledge graphs. Proceedings of the 35th IEEE International Conference on Data Engineering; 2019. p.338-349.
[9] Feddoul L. Semantics-driven keyword search over knowledge graphs. Proceedings of the 19th International Semantic Web Conference; 2020. p.17-24.
[10] Lin X, Ma Z, Yan L. RDF keyword search using a type-based summary. Journal of Information Science and Engineering. 2018; 34(2):489-504.
[11] Sinha SB, Lu X, Theodoratos D. Personalized keyword search on large RDF graphs based on pattern graph similarity. Proceedings of the 22nd International Database Engineering & Applications Symposium; 2018. p.12-21.
[12] Yan W, Ding Y. RDF knowledge graph keyword type search using frequent patterns. Journal of Intelligent & Fuzzy Systems. 2021; 41(1):2239-2253.
[13] Gupta S, Pandey K, Yadav J, Sharma R. Keystroke dynamics based authentication system with unrestricted data collection. Proceedings of the 10th International Conference on Contemporary Computing; 2017. p.1-6.
[14] Yadav J, Pandey K, Gupta S, Sharma R. Keystroke dynamics based authentication using fuzzy logic. Proceedings of the 10th International Conference on Contemporary Computing; 2017. p.1-6.
[15] Ferré S. Analytical queries on vanilla RDF graphs with a guided query builder approach. Proceedings of the 14th International Conference on Flexible Query Answering Systems; 2021. p.41-53
[16] Zafar H, Dubey M, Lehmann J, Demidova E. IQA: Interactive query construction in semantic question answering systems. Journal of Web Semantics. 2020; 64:100586.
[17] Papadaki ME, Spyratos N, Tzitzikas Y. Towards interactive analytics over RDF graphs. Algorithms. 2021;14(2):1-22.
[18] Hu X, Duan J, Dang D. Scalable aggregate keyword query over knowledge graph. Future Generation Computer Systems. 2020; 107:588-600.