Knowledge-driven SPARQL Query Construction Based on Feedback Mechanism

Jing Hu¹², Mei Zhang¹² and Zhi Zhang¹²
¹ College of Computer Science and Technology;
² Big Data Lab; Wuhan University of Science and Technology, Wuhan 430065, China, and Technology, China
Email: bourne0212@qq.com

Abstract. According to the survey, the performance of the existing knowledge graph answering system in entity linking, relation linking and SPARQL query construction in terms of execution time and accuracy cannot meet the requirements of knowledge graph answering system. For this challenge, a new feedback mechanism based Knowledge-Driven query construction method is proposed. This method takes the entity set and predicate set in the problem as input, and constructs SPARQL query statements in a knowledge-driven way to solve simple and complex problems, and further proposes heuristic ideas to deal with implicit entity problems. At the same time, the method also proposes to feed back the query results of the knowledge graph to the entity link and relationship connection steps, so that the SPARQL query statement is optimized again. The evaluation results of the LC_QuAD data show that this method outperform the existing state of the art in precision and recall rate.

1. Introduction
SPARQL query language is a W3C standard, often used to retrieve knowledge base, such as the DBpedia[1], Freebase[2], Yago[3], and Wikidata[4]. Knowledge Graph Based Question Answering System[5] (like Qanary[6]) usually includes entity links, relationship links and SPARQL query construction. Among them, SPARQL query construction tools relies on the results of entity links and relationship links. The entity link tools commonly used in the knowledge graph based question answering system[7] are DBpedia Spotlight[8], AGDISTIS[9], and TagMe[10], and the relationship link tools are ReMatch[11], SIBKB[12], and SPARQL query construction tools are SINA[13], NLIWOD[14]. the Most of these commonly used SPARQL query construction tools need to fully arrange the entities and predicates, and their search space will increase exponentially with the increase in the number of entities and predicates, and consume huge storage space. In addition, it takes a long time to find the correct answer in the full array space of entities and predicates, and it requires additional design of sorting algorithms and pruning algorithms to sort and filter candidate answers, which will cause a poor user experience and the final accuracy rate is not high.
Based on this, this paper proposes a more efficient and accurate SPARQL query construction method (FMKDQ):
- Based on knowledge-driven, with the help of knowledge graph sub-graphs to quickly screen out the correct SPARQL query statements, There is no need to permute entities and predicates.
- No additional design of pruning and sorting algorithms.
- Use the heuristic idea and use the execution result of the previous query to restrict the implicit entities to handle complex problems, so as to find the final query.
2. Related Knowledge

2.1. SPARQL Query Construction
Define a triple $T = \langle e_1, p, e_2 \rangle$ where $e_1$ and $e_2$ are entities and $p$ is a predicate. SPARQL query statements are composed of several triples, and SPARQL query construction is a process of organizing entities and predicates into a series of triples. Among them, for those pending entities and predicates, parameters such as $?E$ and $?P$ are used instead.

2.2. Problem Classification
In order to better construct SPARQL queries, this article divides the problems into three categories, which are fact problems, simple problems, and complex problems.

- A fact problem consists of two entities and a predicate. It is derived from the syntax of "ask where" in the SPARQL query language. It is used to determine whether a question is fact. The answer to the question is "True" or "False".
- A simple problem contains only one entity and one predicate. For example, "Name whose youth club was FC Barcelona?" Contains only one entity {<http://dbpedia.org/resource/FC_Barcelona>} and a predicate {<http://dbpedia.org/property/youthclubs>}. The number of entities in the problem is equal to or greater than the number of predicates in the problem. Conversely, if the number of entities is less than the number of predicates, it is an implicit entity problem.

3. Knowledge-driven Query Construction
The query construction process in this article reuses the results of entity links and relationship links, which extracted the set of predicates and entity from the question as input, and output the SPARQL query corresponding to the problem. For different problem types, FMKDQ will design different building processes.

3.1. Query Construction Process for Fact Problems
For the fact problem Q1, from the collection of entities randomly selects an entity $E_1$, it constructs the candidate query triples with the predicate $P$, then use the algorithm choose the correct query $S_1$. Because the fact problem only contains a triple, execute $S_1$ directly and you will get the result list $R$. In order to confirm that the answer to the factual question is "True" or "False", it is only necessary to determine whether another entity $E_2$ in the entity set $E$ meet $E_2 \in R$. If yes, the answer is "True", otherwise it is "False". If the correct query cannot be found, it is determined that the result of the input entity link and relationship link is wrong, and the candidate entity set and predicate set need to be reconstructed.

3.2. Query Construction Process for Simple Problems
For the simple problem $Q_2$, from the collection of entities randomly selects an entity $E$, it constructs the candidate query triples with the predicate $P$, then use the algorithm choose the correct query $S$. Since the simple problem has only one two-tuple, execute $S$ directly to get the result set $R$, then the answer to the simple problem $Q_2$ is the result set $R$. Similarly, if you cannot find the correct query, you need to reconstruct the candidate entity set and predicate set.

3.3. Query Construction Process for Complex Problems
According to a survey on the large-scale complex question answering system data set LC-QuAD[15], it is found that there are no more than two predicates in this data set. Therefore, FMKDQ only considers the case of two predicates. At the same time, in order to better handle the query construction process, implicit entity problems and explicit entity problems in complex problems are processed separately.
3.3.1. Query construction process for explicit entity problems. As shown in Figure 1, the entity e1 is randomly selected from the entity set, and then the entity e1 can be used to piece together two candidate triple query statements:

c1: "select distinct ?p1 where {e1 ?p1 ?x}"
c2: "select distinct ?p2 where {?x ?p2 e1}"

![Diagram](image.png)

**Figure 1.** Explicit entity problem construction process

Execute c1 and c2 to obtain the result sets RC1 and RC2, respectively, and determine whether the predicate p1 exists, which can satisfy $p1 \in P$. Under the above conditions, if p1 satisfies $p1 \in RC1$, then ?p1 in c1 is replaced by p1, and if p1 satisfies $p1 \in RC2$, then ?p2 in c2 is replaced by p1, and SPARQL query s1 is obtained. The entity e2 and the predicate p2 are combined to obtain the SPARQL query s2. Execute s1 and s2 to get the result sets R1 and R2. The intersection R of R1 and R2 is the answer to problem Q3. Similarly, if you cannot find the correct query, you need to reconstruct the candidate entity set and predicate set.

For example, the explicit entity problem Q3 "List TV shows with producer as Erik Bork and company is Dream Works Television?", The entity set E\{<http://dbpedia.org/resource/Erik_Bork>, <http://dbpedia.org/resource/DreamWorks_Television>\} and predicate set P\{<http://dbpedia.org/ontology/producer>, <http://dbpedia.org/ontology/company>\} . Define the entity <http://dbpedia.org/resource/Erik_Bork> as e1, and then use this entity to piece together two candidate triple query statements:

c1 : "select distinct p1 where {<http://dbpedia.org/resource/Erik_Bork> ?p1 ?x}"  
c2 : "select distinct p2 where {?x ?p2 <http://dbpedia.org/resource/Erik_Bork>}"

Execute c1 and c2 to get the result sets RC1 and RC2 respectively. Then find the predicate p1 <http://dbpedia.org/ontology/producer> meet $p1 \in RC2$, then ?p2 in c2 is replaced by p1 to obtain the first SPARQL query s1, as query table 1 shown.

| Table 1. SPARQL query statement s1 |
|-----------------------------------|
| select distinct ?x where { |
| ?x <http://dbpedia.org/ontology/producer> |
| <http://dbpedia.org/resource/Erik_Bork> |
| } |
Then execute s1 to get the result set R1, as shown in Table 2.

**Table 2. Result set r1**

```
['http://dbpedia.org/resource/Band_of_Brothers_(miniseries)',
'http://dbpedia.org/resource/UC:_Undercover'
]
```

Then combine another entity e2<http://dbpedia.org/resource/DreamWorks_Television> and another predicate p2<http://dbpedia.org/ontology/company> to obtain a SPARQL query s2, such as query statement Table 3 shown.

**Table 3. SPARQL query statement s2**

```sparql
select distinct ?x where{
  ?x <http://dbpedia.org/ontology/company>
  <http://dbpedia.org/resource/DreamWorks_Television>
}
```

Execute s2 to get the result R2. The result set R of R1 and R2 is ['http://dbpedia.org/resource/Band_of_Brothers_(miniseries)']. So explicit entity problem Q3 "List TV Shows with Producer Erik Bork and Company AS IS Dream Works Television?" The answer is the result set R.

3.3.2. **Query construction process for implicit entity problems.** As shown in Figure 2, first use entity e to piece together two candidate triple query statements:
c1 : "select distinct ?p1 where {e ?p1 ?x}"
c2 : "select distinct ?p2 where {?x ?p2 e}"

**Figure 2. Implicit entity problem construction process**

Execute c1 and c2 to obtain the result sets RC1 and RC2, respectively, and determine whether the predicate p1 exists, which can satisfy p1∈P. Under the above conditions, if p1 satisfies p1∈RC1, then ?p1 in c1 is replaced by p1, and if p1 satisfies p1∈RC2, then ?p2 in c2 is replaced by p1, and SPARQL query s1 is obtained. Execute s1 to get the result set R1, and then use the result set R1 to get s2. Because the result set R1 can help constrain the scope of implicit entities. Therefore, the result set R1 can be defined as e2, and e2 can be combined with another predicate p2 to obtain SPARQL query s2. Execute s2 to get the result set R2, which is the answer to question Q4.
4. Experimental Evaluation

4.1. Data Set
This experiment uses LC-QuAD as the experimental corpus. This corpus has 5000 question-answer pairs (questions and SPARQL query statements corresponding to the questions) based on the DBpedia knowledge base, and the entity set and predicate set extracted from the question. Concentrated problems cover all the problems types mentioned in this article, that is, it include fact problems, simple problems, and complex problems, and most of them are complex problems that are difficult to answer. They are representative data to test the system's ability to handle complex questions. Set one.

4.2. Experimental Settings
This experiment uses DBpedia as a knowledge base and query site, and randomly selected 200 questions from LC-QUAD as the test set. These 200 questions contain 40 fact problems, 80 simple problems, and 80 complex problems. In this experiment, the steps of entity link and relationship link are omitted, and the results of entity link and relationship link, which means, entities and predicates are used as input, and SPARQL query statements corresponding to the problem are output. In addition to verifying the effect of FMKDQ, this experiment also compares FMKDQ with existing query construction systems SINA and NLIWOD in Precision, Recall and F-score.

At the same time, in order to verify the effectiveness of this method and reflect the characteristics of its feedback mechanism, this article also applies FMKDQ to Qanary, that is, combining the entity link and relationship link and query construction to calculate the overall performance of the Qanary graph question answering system. The accuracy, recall rate and F-value of the proposed method are compared with those without the method to verify the effectiveness of the proposed method.

4.3. Experimental Indicators
Each query construction method is evaluated using the following metrics:
- Micro Precision: referred to as MP. For each question, the correct query statement given by the SPARQL query construction tool meet the actual correct query statement then mark MP as 1, otherwise as 0.
- Precision: referred to as P. the comprehensive average of the MP constructed by each SPARQL query construction tool.
- Micro Recall: referred to as MR. For each question, the correct query sentence given by the SPARQL query construction tool meet the actual correct query sentence then mark MR as 1, otherwise as 0.
- Recall: referred to as R. the comprehensive average of the MR of each SPARQL query construction tool.
- Micro F-score: referred to as MF, for each question, query the F-value given by the SPARQL query construction tool, that is:
  \[ F = \frac{2 \times MP \times MR}{MP + MR} \]
- F-score: referred to as F, the comprehensive average value of the MF of each SPARQL query construction tool.

4.4. Reviews
In order to verify the FMKDQ performance, this experiment tested 40 fact problems, 80 simple problems and 80 complex problems, as shown in Table 4, Table 5, Table 6. For the three types of problem, the Precision, Recall and F-score of FM KDQ can reach more than 0.8, far exceeding SINA and NLIWOD, especially for complex problems. Significant advantages over the other two systems.
Table 4. Effect of fact problems query-building

| name | SINA   | NLIWOD | FMKDQ |
|------|--------|--------|-------|
| P    | 0.475  | 0.65   | 0.8   |
| R    | 0.475  | 0.65   | 0.8   |
| F    | 0.475  | 0.65   | 0.8   |

Table 5. Effect of simple problems query-building

| name | SINA   | NLIWOD | FMKDQ |
|------|--------|--------|-------|
| P    | 0.6125 | 0.75   | 0.8875|
| R    | 0.6125 | 0.75   | 0.8875|
| F    | 0.6125 | 0.75   | 0.8875|

Table 6. Effect of complex problems query-building

| Name  | SINA   | NLIWOD | FMKDQ |
|-------|--------|--------|-------|
| P     | 0.225  | 0.475  | 0.85  |
| R     | 0.225  | 0.475  | 0.85  |
| F     | 0.225  | 0.475  | 0.85  |

Table 7. Comprehensive performance of each method on the LC-QuAD corpus

| System Name | Number of questions | P       | R       | F       |
|-------------|---------------------|---------|---------|---------|
| SINA        | 200                 | 43.3%   | 45.0%   | 44.1%   |
| NLIWOD      | 200                 | 62.0%   | 64.0%   | 63.0%   |
| FMKDQ       | 200                 | 85.5%   | 85.5%   | 85.5%   |

Table 7 as shown, the FMKDQ precision can be reached almost 0.85, F-score can reach 0.85 or more, with respect to the SINA ( F is 0.44 ) and NLIWOD ( F is 0.63 ) has a significant Promotion.

4.5. Conclusion
This paper proposes a new feedback mechanism based Knowledge-Driven query construction method (FMKDQ). It uses a knowledge-driven method to help determine SPARQL query statements, solves implicit entity problems with the help of heuristic idea, provide feedback on entity links steps and relationship link steps based on the results of query construction. By using LC-QuAD data set FMKDQ evaluation, found whether it's facing simple or complex problems, the search space and run time has been significantly improved.

However, FMKDQ still has many areas that need improvement and further research. This method is only for query construction of up to two predicate problems, and the method will be extended to problems that can handle more predicates. In addition, even if the query construction method proposed in this article is combined, the performance of the graph question answering system on the LC_QuAD data set has a lot of room for improvement, indicating that the graph question answering system still needs much improvement, and the graph question answering system will continue to be optimized in the future.
5. References

[1] Lehmann J, Isele R, Jakob M, et al. DBpedia—a large-scale, multilingual knowledge base extracted from Wikipedia[J]. Semantic Web, 2015, 6(2): 167-195.

[2] Yao X, Berant J, Van Durme B. Freebase qa: Information extraction or semantic parsing?[C]//Proceedings of the ACL 2014 Workshop on Semantic Parsing. 2014: 82-86.

[3] Rebele T, Suchanek F, Hoffart J, et al. YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames[C] //International Semantic Web Conference. Springer, Cham, 2016: 177-185.

[4] Vrandečić, Denny, Krötzsch, Markus. Wikidata: a free collaborative knowledgebase[J]. Communications of the Acm, 2014, 57(10):78-85.

[5] Lukovnikov D, Fischer A, Lehmann J, et al. Neural network-based question answering over knowledge graphs on word and character level[C]//Proceedings of the 26th international conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2017: 1211-1220.

[6] Both A, Diefenbach D, Singh K, et al. Qanary—a methodology for vocabulary-driven open question answering systems[C]//European Semantic Web Conference. Springer, Cham, 2016: 625-641.

[7] Singh K, Both A, Sethupat A, et al. Frankenstein: A Platform Enabling Reuse of Question Answering Components[C]//European Semantic Web Conference. Springer, Cham, 2018: 624-638.

[8] Mendes P N, Jakob M, Garcia-Silva A, et al. DBpedia spotlight: shedding light on the web of documents[C]//Proceedings of the 7th international conference on semantic systems. ACM, 2011: 1-8.

[9] Usbeck R, Ngomo A C N, Röder M, et al. AGDISTIS-graph-based disambiguation of named entities using linked data[C]//International semantic web conference. Springer, Cham, 2014: 457-471.

[10] Ferragina P, Scaiella U. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities)[C]//Proceedings of the 19th ACM international conference on Information and knowledge management. ACM, 2010: 1625-1628.

[11] Mulang I O, Singh K, Orlandi F. Matching natural language relations to knowledge graph properties for question answering[C]//Proceedings of the 13th International Conference on Semantic Systems. ACM, 2017: 89-96.

[12] Singh K, Mulang I O, Lytra I, et al. Capturing knowledge in semantically-typed relational patterns to enhance relation linking[C]//Proceedings of the Knowledge Capture Conference. ACM, 2017: 31.

[13] Shekarpour S, Auer S. SINA: semantic interpretation of user queries for question answering on interlinked data by Saeedeh Shekarpour with Prateek Jain as coordinator[J]. ACM SIGWEB Newsletter, 2014 (Summer): 30(Summer):1-1.

[14] Saleem M, Dastjerdi S N, Usbeck R, et al. Question Answering Over Linked Data: What is Difficult to Answer? What Affects the F scores?[C]//BLINK/NLIWoD3@ ISWC. 2017.

[15] Trivedi P, Maheshwari G, Dubey M, et al. LC-QuAD: A corpus for complex question answering over knowledge graphs[C]//International Semantic Web Conference. Springer, Cham, 2