Searching Software Knowledge Graph with Question

Min Wang[1,2], Yanzhen Zou[1,2(✉)], Yingkui Cao[1,2], and Bing Xie[1,2]

[1] Key Laboratory of High Confidence Software Technologies, Peking University, Ministry of Education, Beijing 100871, China
{wangmin1994,zouyz,caoyingkui,xiebing}@pku.edu.cn
[2] School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China

Abstract. Researchers have constructed a variety of knowledge repositories/bases in different domains. These knowledge repositories generally use graph database (Neo4j) to manage heterogeneous and widely related domain data, which providing structured query (i.e., Cypher) interfaces. However, it is time-consuming and labor-intensive to construct a structured query especially when the query is very complex or the scale of the knowledge graph is large. This paper presents a natural language question interface for software knowledge graph. It extracts meta-model of software knowledge repository, constructs question related Inference Sub-Graph, then automatically transfers natural language question to structured Cypher query and returns the corresponding answer. We carry out our experiments on two famous open source software projects, build their knowledge graphs and verify our approach can accurately answer almost all the questions on the corresponding knowledge graph.

Keywords: Software reuse · Knowledge repository · Knowledge graph · Natural language search · Graph search

1 Introduction

Software reuse is a solution to avoid duplication of effort in software development, which can improve the efficiency and quality of software engineering [1,2]. In recent years, with the rapid development of open source software, a large number of reusable software projects have emerged on the Internet, such as Apache Lucene, Apache POI, jfreeChart, etc. Besides the source code, these software projects often contain different types of natural language text resources, such as user manuals, mailing-list, issue reports, user forum discussions, etc. To utilize and analyze these documents and source code, researchers propose and construct

Supported by the Foundation item: National Key Research and Development Program (2016YFB1000801), National Science Fund for Distinguished Young Scholars (61525201).

X. Peng et al. (Eds.): ICSR 2019, LNCS 11602, pp. 115–131, 2019.
https://doi.org/10.1007/978-3-030-22888-0_9

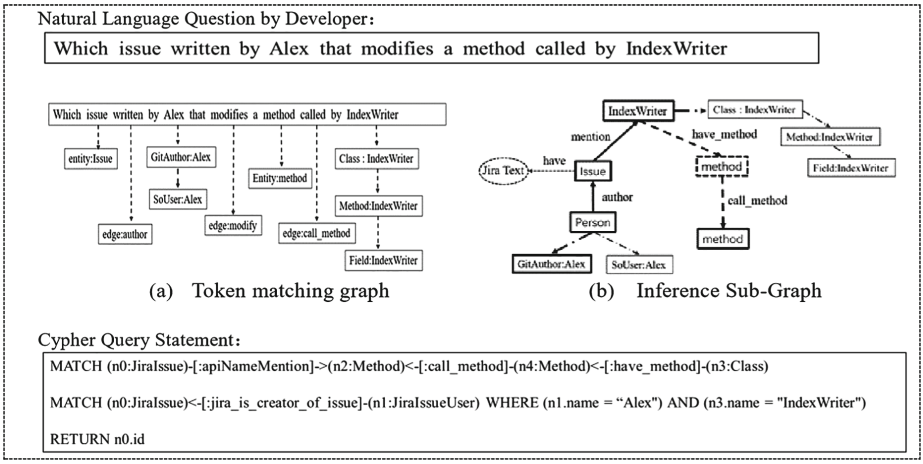

Fig. 1. A Search question and related Inference Sub-Graph (ISG)

various domain-specific software knowledge bases and knowledge graphs. Typically, Peking University proposed the specific-domain software projects knowledge graphs [3]; Fudan university proposed the software projects API knowledge graphs [27], etc.

These knowledge bases/knowledge graphs often utilize Neo4j and other graph database storage, which support formal query (i.e., Cypher). It requires the user be familiar with the Cypher grammar, and manually translate the user intention into a Cypher query statement. Meanwhile, the manual construction of Cypher query statement requires a clear understanding of the meta-model of the knowledge graph, i.e., the types of knowledge entities inside the knowledge graph. When the scale of meta-model becoming is very large, the cost of constructing Cypher query is very large too. Therefore, it is necessary to provide a knowledge base/knowledge graph query interface based on natural language, which can automatically transform users' natural language questions into formal query statements.

To address this problem, existing work in the process of natural language question parsing often rely on the large general domain knowledge base on the Internet (such as WordNet, DBpedia, Freebase, Yago, etc.), the calculation is very complex [6]. Typically, Percy Liang et al. utilized Lambda DCS (Dependency Based Composition Semantics) to get a logical representation of a natural language question and constructed a formal query for knowledge base or knowledge graph based on its logical representation. However, all these methods face the challenge of precise transformation of natural language descriptions. In our research, we found that the concepts in the software knowledge graph are quite different from those in the general open-domain knowledge base due to the characteristics of the specific-domain software. Therefore, it is not important that matching the specific predicate phrases. In contrast, we should pay

more attention to the precise analysis of the overall semantics of the question. Figure 1 presents a natural language question from a developer and its corresponding construction process of structured Cypher query statement. When software developers reuse Apache Lucene's open source projects, they raise a query like this: *"Which issue written by Alex that modifies a method called by IndexWriter?"*, the key words included in this statement are *issue, Alex, modify, method, call, IndexWriter*, etc. Each key word has several concepts on the knowledge graph to correspond to it. *Alex*, for example, can be matched to a Person entity with a name attribute on the knowledge graph, but the knowledge graph has two similar conceptual entities, i.e., *GitAuthor: Alex developers* and *SoUser: Alex (stackOverflow users)*. Similarly, the *IndexWriter* can be matched to a software project source code knowledge of an entity on the knowledge graph, but it could match to *Class*, *Method*, and *Field*, three levels of entities. So what are the concepts that developers are trying to convey? Which concepts correspond to a knowledge graph entity? In order to address the problem of precise matching between natural language tokens and knowledge graph entities, we propose a method of reasoning and locating sub-graph in the knowledge graph corresponding to natural language questions, which is called Inference Sub-Graph. Inference Sub-Graph is an intermediate bridge from natural language questions to Cypher query statements: the generation of Inference Sub-Graph requires in-depth analysis and the use of knowledge graph information and knowledge graph meta-model to "match" the semantics in natural language questions.

Based on this insight, this paper proposes a natural language query approach on the software knowledge graph. It could transfer natural language question to structured Cypher query precisely through extracting software knowledge graph meta-model and constructing Inference Sub-Graph, then display the corresponding answer on the knowledge graph. Compared with the existing work, the main contributions of this work are:

- We propose a novel approach for transferring natural language question to formal Cypher query. Different from traditional semantic parsing, this algorithm generates an Inference Sub-Graph as a transformation bridge between natural language and formal query that can more accurately express the deep meaning of natural language. On the other hand, the ambiguity problem of natural language can be solved remarkably by measuring the Inference Sub-Graph;
- We implemented a natural language question interface and validated our approach on the software project knowledge graph of Apache Lucene and Apache Nutch's open source project. For each knowledge graph, we provided 66 real natural language questions in the process of software reuse, and we can achieve an accuracy of 93.9%.

2 Overview

We propose and implement a natural language question answering approach and system on software knowledge graph. In our approach, we firstly propose the concept of Inference Sub-Graph, and build a transformation bridge between natural

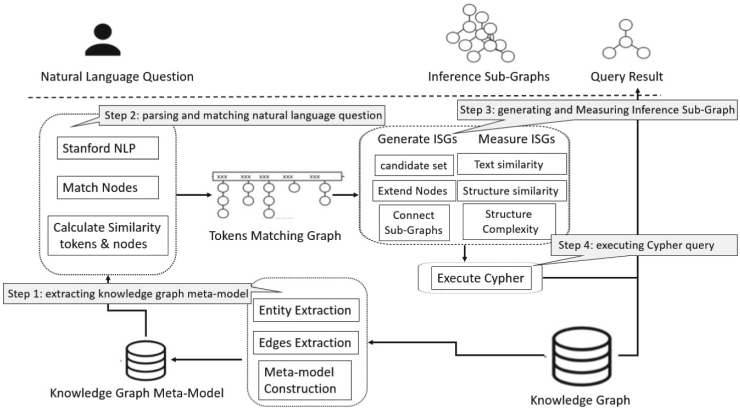

Fig. 2. The workflow of natural language Q&A solution

language questions and formal query statements by generating and measuring Inference Sub-Graph. On the one hand, compared with the traditional logical formal system of semantic parsing, the approach of sub-graph transformation [24,25] is easier to express the semantic information of natural language. On the other hand, the generation and measurement of Inference Sub-Graphs can solve the ambiguity problem of natural language [25]. Figure 2 shows the workflow of our approach in this paper. Our approach is mainly composed of four parts: (1) extracting knowledge graph meta-model; (2) parsing and matching of natural language questions; (3) generating and measuring of Inference Sub-Graph; (4) constructing Cypher query statement.

2.1 Extracting Software Project Knowledge Graph Meta-Model

In order to adapt various Software Project Knowledge Graph, firstly, we define some concepts as follows:

(1) Entity-Type: each entity in the knowledge graph has a type. It defined as: {Entity-Type} = {Entity.type}.
(2) Relation-Type: each type of relationship in the knowledge graph is determined by the name of the relationship itself and the type of entity at both ends of the relationship. It defined as: {Relation-Type} = {(Relation.start.type, Relation.end.type, Relation.name)}.
(3) Attribute-Type: The type of each attribute in the knowledge graph is determined by its associated entity type and attribute name. It defined as: {Attribute-Type} = {(Attribute.entity.type, Attribute.name)}.
(4) meta-model: A schema graph with Entity-Type as the node, Relation-Type as the edge, and Attribute-Type as the property of the Entity-Type. It defined as: meta-model = {Entity-Type},{Relation-Type}.

Based on the above concepts, we traverse the software project knowledge graph, store all the entities in it and record all entity types as meta-model's entities;

then we traverse the software project knowledge graph, store all the relations in it and record all relation types as meta-model's relations.

2.2 Parsing and Matching Natural Language Question

In this subsection, we utilize Stanford NLP tools to parse natural language questions, including token segment, part-of-speech and filter stop-words, and then we match these tokens with elements from knowledge graph.

We propose a priority-based matching algorithm, i.e., hierarchically matching each tokens (i.e., parsing from natural language question) with elements from knowledge graph or knowledge graph meta-model. Each token may be matched to any elements about the knowledge graph, including: entity, entity type, attribute, attribute type, and relation, relation type, etc. We manually construct a synonym table based on software engineering specific-domain knowledge.

For example, the natural language question is *"when the class that call the method updatependingmerge is changed by Yonik Seeley ?"* Obviously, we can match *class* with one of entity type in the knowledge graph, i.e., Class, similarly, we can match *updatependingmerge* with one of entity in the knowledge graph, i.e., method named *updatependingmerge*. However, we can't directly match any elements with *call* and *changed*, so we utilize synonym table to match some synonym (e.g., *call* match with the relationship "call-method", *changed* match with the relationship "commit", etc.)

Obviously, it's possible that a token matches to more than one elements. To address this problem, firstly, we utilize Minimum Edit Distance algorithm to calculate the similarity between tokens and elements, so we can filter some irrelevant elements. However, we still retain some ambiguous matching pairs, we can eliminate this ambiguity in the latter approach. Finally, we obtain a {tokens,elements} matching graph, as shown in Fig. 1.

2.3 Generating and Measuring Inference Sub-Graph

Due to the ambiguity and complexity of natural language, tokens based on natural language processing may match to different knowledge graph elements, thus we obtain candidate sub-graphs with different semantics. In order to guarantee the quality of candidate sub-graphs, we propose an Inference Sub-Graph generation algorithm by extending hidden nodes and hidden edges, and calculate the feature values of each candidate Inference Sub-Graph. The measurement features include three aspects: textual similarity, structural similarity and Inference Sub-Graph structural complexity, and then we recommend the optimal Inference Sub-Graph to developers according to the measurement results. In this process, on the one hand, we construct the intermediate transformation bridge between natural language and formal query statement, on the other hand, we resolve the ambiguity problem of natural language by measuring the optimal sub-graph. Finally, we transform the optimal Inference Sub-Graph into a Cypher query statement and return the query result. In order to prevent possible semantic

understanding deviation, we also support the visualization of the Inference Sub-Graph in the interactive process.

In the next section, we will introduce the details about the generation and measurement of Inference Sub-Graphs.

2.4 Constructing Cypher Query Statement

Here, we transform the Inference Sub-Graph into a Cypher query statement. The process of constructing a Cypher can be divided into three parts, corresponding to Match, Where and Return statement respectively.

- Match statement corresponds to the graph structure of Inference Sub-Graph. However, Inference Sub-Graph expresses a two-dimensional structure information, a Match statement only describes a one-dimensional structure information, namely a chain. So we adopt the heuristic method of dividing the path on the Inference Sub-Graph into a number of chain and set.
- Where statement corresponds to the all attribute values of nodes in the Inference Sub-Graph, each attribute value is the filter conditions of an entity, so attributes are added to the Where statement.
- Return statement is determined by the returned node properties. The returned node properties are determined by the interrogative in the question.

3 Generating and Measuring Inference Sub-Graph

In Sect. 2.2, after parsing natural language questions, their tokens may match to multiple knowledge graph elements. In order to ensure the correctness of the overall sentence meaning, we propose the concept of Inference Sub-Graph, and give the corresponding generation and measurement algorithm of Inference Sub-Graph.

Inference Sub-Graph: An Inference Sub-Graph is a sub graph in the knowledge graph, including: entities and edges, this sub-graph is inferred by the {tokens,elements} pairs from natural language question and knowledge graph, and so this sub-graph contains similar semantics with the natural language questions.

3.1 Generating Inference Sub-Graph

Inference Sub-Graph is a sub-graph on the knowledge graph meta-model. Due to the ambiguity of natural language, the semantics of natural language is often incomplete. As shown in Fig. 1, in the example of natural language question *"Which issue written by Alex that modifies a method called by IndexWriter"*, we need to extend a hidden method entity node (i.e., method) and a hidden association relation (i.e., have_method) between the entity class (i.e., IndexWriter) and the entity method called by IndexWriter. i.e., the complete semantics of the

natural language question should be: a method is modified because of an issue proposed by Alex, and the method is called by a method member method in the class IndexWriter.

To address this problem, we propose a novel approach, which is called Inference Sub-Graph. Algorithm 1 illustrates the process of generating Inference Sub-Graph. The input is a set of disconnected sub-graphs, which extracting from knowledge graph according the {tokens,elements} matching graph (Sect. 2.2). The output is a set of connected Inference Sub-Graphs. Firstly, as shown in algorithm in line 2–6, by parsing natural language questions and matching elements in the knowledge graph, we obtain a matching graph. Due to the ambiguity of natural language, each natural language token may correspond to multiple knowledge graph related elements. We retain these ambiguities in the previous process. We utilize a Bean-Search algorithm to connect all possible {tokens,elements} matching graph, and so we can obtain a set of candidate disconnected sub-graph, i.e., $S_{disconnect} = G_{disconnect}$. Each set represents a possibility of the knowledge graph element set corresponding to the natural language question. Then, in line 7–11 in algorithm 1, we extend the hidden nodes of the disconnected sub-graphs by inferring the knowledge graph, and we get the graphs with hidden nodes and hidden edges, i.e., $S_{hidden} = G_{hidden}$. Finally, in line 12–17 in algorithm 1, we utilize a Minimum Spanning Tree algorithm to connect these disconnected sub-graphs, so we obtain a set of connected Inference Sub-Graphs, i.e., $S_{connect}$.

Algorithm 1. Inference Sub-Graph Generation

Input: A set of disconnected Sub-Graphs : $S_{disconnect}$

Output: A set of candidate connected Sub-Graphs : $S_{connect}$

```
1.  Function Connect(Sdisconnect)
2.      Shidden := {}
3.      For each (Gdisconnect in Sdisconnect)
4.          Ssub-hidden :=getG-hidden(Gdisconnect)
5.          Shidden := Shidden ∪ Ssub-hidden
6.      End for
7.      Sforest := {}
8.      For each (Ghidden in Shidden)
9.          Ssub-forest := getG-forest (Ghidden)
10.         Sforest = Sforest ∪ Ssub-forest
11.     End for
12.     Sconnect := {}
13.     For each (Gforest in Sforest)
14.         Ssub-connect := getG-connect (Gforest)
15.         Sconnect = Sconnect ∪ Ssub-connect
16.     End for
17.     Return Sconnect
18. End Function
```

3.2 Measuring Inference Sub-Graphs

In the above steps, we generate a set of Inference Sub-Graphs, we have to select the optimal Inference Sub-Graph as a bridge with a Cypher query statement.

We propose a series of heuristic rules to calculate some measured features as follows:

(1) Textural Similarity between Inference Sub-Graph and NL-Question

In Sect. 2.2, as the Fig. 1 shows, we retain some candidate elements which match the tokens, i.e., each token has a candidate elements list, each element in this list has a similar rank with tokens, so, we can calculate the similarity between Inference Sub-Graph and Natural Language Question as follows:

$$Score_{T-similar} = \omega_{T-similar} \times \sum_{t \epsilon token} (1 - \frac{t.mapping.rank}{k}) \quad (1)$$

In the above formula, t denotes the tokens from the parsed natural language questions (Sect. 4.2), $t.mapping.rank$ denotes the element's rank in the candidate list, k denotes the number of all elements in the matching graph, and $\omega_{T-similar}$ denotes the weight of textural similarity in the feature set, we can manually assign a weight value.

(2) Structural Similarity between Inference Sub-Graph and NL-Question

Calculating the structural similarity between the Inference Sub-Graph and the natural language question, we consider the difference between the relative distance between the pair of nodes in the Inference Sub-Graph and the relative distance of the corresponding words in the natural language question as the metric factor. We can calculate the structural similarity as follows:

$$\begin{aligned} Score_{S-similar} = \omega_{S-similar} \times (& \sum_{i \epsilon V(G)} \sum_{j \epsilon outVertex(i)} (Position_i - Position_j) \\ & + \sum_{i \epsilon V(G)} \sum_{e \epsilon outEdge(i)} (Position_e - Position_i + token_e.direct)) \end{aligned} \quad (2)$$

In the above formula, $Position$ denotes the position of the natural language token in the question corresponding to the node or edge in each sub-graph. $outVertex(i)$ denotes $node_i$'s adjacent node set. $outEdge(i)$ denotes $node_i$'s outgoing edge. $token_e.direct$ denotes tense of natural language token corresponding to the direction of edge e. $\omega_{S-similar}$ denotes the weight of structural similarity in the feature set.

(3) Structural Complexity of the Inference Sub-Graph

In the process of generating the Inference Sub-Graph, we extend the hidden nodes of the matching graph, which leads to the change of the structure of the Inference Sub-Graph. Considering the complexity of this structural change and the connection between natural language questions, we calculate the structural complexity of the Inference Sub-Graph as one of the measured features:

$$Score_{complex} = \omega_{complex} \times (|E(G)| + |v(G)| - |V(G_{hidden})| - |E(G_{hidden})|) \quad (3)$$

In the above formula, $E(G)$ and $V(G)$ respectively denotes original number of the nodes and edges in the Inference Sub-Graph. $V(G_{hidden})$ and $E(G_{hidden})$ respectively denotes the number of hidden nodes and edges that we extend. $\omega_{complex}$ denotes the weight of structural complexity in the feature set.

Finally, we calculate these feature values to measure a Inference Sub-Graph, and so we can obtain a optimal Inference Sub-Graph. On the one hand, it

resolves the gap between natural language and formal query, and on the other hand, it resolves the ambiguity when natural language matches knowledge graph elements.

4 Experiment and Evaluation

Based on the above solution, we designed and implemented a natural language question answering system for software project knowledge graph, which is called SnowSearch (**S**oftware K**now**ledge **Search**).

Our experiments are used to answer the following research questions:

RQ1: Does SnowSearch effectively answer natural language questions?

We are concerned about whether our approach could effectively return the correct query results when the developers input a natural language question.

RQ2: Does our Inference Sub-Graph solve the transformation between natural language and Cypher query?

We propose a approach of Inference Sub-Graph on the knowledge graph to serve as a bridge between natural language and formal query statements. We are concerned about whether the Inference Sub-Graph could express natural language more effectively and accurately.

RQ3: Are the three metric features of the Inference Sub-Graph reasonable?

We propose three metrics of textural similarity, structural similarity and Inference Sub-Graph complexity to measure the Inference Sub-Graph, so as to obtain the optimal Sub-Graph. We are concerned about whether the three metric features proposed in this paper are reasonable.

4.1 Software Project Knowledge Graph

Our work is based on software project knowledge graph proposed by Lin et al. [3,26]. Software knowledge graph is defined as a graph for representing relevant knowledge in software domains, projects, and systems. In a software knowledge graph, nodes represent software knowledge entities (e.g., classes, issue reports and business concepts), and directed edges represent various relationships between these entities (e.g., method invocation and traceability link). Based on their tools, we can automatically construct a software project knowledge graph.

To evaluate our approach, we constructed knowledge graph of two famous open source software projects, Apache Lucene and Apache Nutch. The open source project Lucene is a Java-implemented information retrieval program library; the open source project Nutch is a Java-implemented Open source search engine. We collected a large number of multi-source heterogeneous data on Lucene and Nutch on the Internet, including software source code, mailing-lists, issue reports, and StackOverflow posts. The statistics of related resources

Table 1. Software resources statistics in Apache Lucene and Nutch

Project	Source Code		Documents			Knowledge Graph		Time min
	version	Code File	Issue Report	Mailing-list	StackOverflow Post	Entities	Edges	
lucene	6.3.0	1941	7439	15769	9376	39,419	176,3694	28
nutch	1.5.1	553	1254	2759	947	13,861	8,4908	7

Table 2. Meta-Model of Software Project Knowledge Graph

Data Type	Knowledge Entity	Relation Type
Source code	Class, Interface, Field, Method	extends, implements, dependency, call, include, type, method parameters, return, throw
Issue Report	Issue report, patches, comments, jira user	Issue-patches, Issue-Comments
Mailing-list	mail content, mail user	mail-send, mail-reply
StackOverflow Posts	questions, answers, comments, SoUser	Question-proposed, answer-question, answer-received

are shown in Table 1. For example, the open source project Lucene is a well-known information retrieval library implemented by the Java language. We collected more than 8 GB of Lucene project data on the Internet, including 67 Versions, more than 800,000 lines of source code, 244,000 e-mails, more than 7,500 issue reports, and a large number of related StackOverflow posts. Finally, we constructed Apache Lucene's knowledge graph, and statistics of the graph are shown in Table 2.

4.2 Question Example

We invited four developers who are familiar with Apache Lucene and Nutch to ask some natural language questions related to these two projects. These questions mainly cover factual questions such as *Who, What, When,* and the *List*. Finally, the four developers proposed 66 questions about the Lucene project and 66 questions about the Nutch project. As shown in Table 3, we randomly select 22 sample questions that developers ask for Lucene open source projects. Among them, the number of entities in the Inference Sub-Graph corresponding to *What/Which* type, *Who* type and *When* type questions is about 4 on average, and the number of edges in Inference Sub-Graph is about 3 on average. The number of entities in Inference Sub-Graph corresponding to the questions of *List* type is about 97 on average, and the number of edges in Inference Sub-Graph is

Table 3. 22 questions used to search on Lucene knowledge graph

Type	Natural Language Questions	Entity Num	Edge Num
What/Which	what's the superclass of the Class that is extended by the Class call updatependingmerge	5	3
	which class have updatependingmerge	2	1
	which class call updatependingmerge	2	1
	Which commit is about "custom MergeScheduler implementation"	4	3
Who	Who change IndexWriter	3	2
	who change IndexWriter and change IndexReader	5	4
	who mention a method called by IndexWriter	4	3
	Who write a commit about "custom MergeScheduler implementation"	4	3
	who write a jiraissue about "custom MergeScheduler implementation"	4	3
	who write a answer about IndexWriter	3	2
When	When a commit about "custom MergeScheduler implementation" is written	4	3
	When Dave Kor send a mail about IndexWriter	3	2
	When is IndexWriter changed	3	2
	When IndexWriter is change by a commit written by Yonik Seeley	4	3
List	list method belong to IndexWriter	53	52
	list field belong to IndexWriter	9	8
	list Class mentioned by Yonik Seeley	20	19
	list method of Class IndexWriter	53	52
	Issue about IndexWriter changed by Alexb	8	7
	list all mail sent by Dave Kor	75	74
	list all JiraIssue that mention IndexWriter	478	477
	id of Issue written by Alexb that mention a method called by Class IndexWriter	76	75

Table 4. The precise results of 132 Q&A about Lucene and Nutch

Project	What/Which		Who		When		List		Amount	
	P@1	P@2	P@1	P@2	P@1	P@2	P@1	P@2	P@1	P@2
lucene	12/12	12/12	17/18	18/18	12/12	12/12	21/24	22/24	62/66	64/66
nutch	14/14	14/14	16/16	16/16	11/12	12/12	21/24	22/24	62/66	64/66

about 96 on average. The data analysis results of 22 examples of natural language questions show that the natural language questions raised by developers in the process of software reuse exist objectively, and have certain reasoning complexity and Q&A difficulty for the knowledge base query.

4.3 RQ1: Q&A Effectiveness Evaluation

We think that only the accuracy of top-1 and top-2 Cypher need to be considered. i.e., the generated correct Cypher query is ranked in the top k position in all possible Cypher candidate sets, thus the question answering result is correct. The experimental results are shown in Table 4. It can be seen that 62 of the 66 questions for the knowledge graph of Lucene project return correct Cypher query in top-1, while 64 questions return correct Cypher query in top-2. The same experimental results were obtained in 66 questions for Nutch project knowledge graph, which indicate that most natural language questions of different software projects can be correctly answered.

The experiment result shows that our approach of natural language question answering on the software project knowledge graph can achieve more than 93.3% accuracy. Considering the natural language grammar of *List* type questions are more complicated, and the corresponding entities in the Inference Sub-Graph are more complicated, natural language question parsing and matching may produce a certain error. For example: In natural language question *"list all JiraIssue that mention IndexWriter"*, because the JiraIssue and IndexWriter get involved with large number of entities, lead to natural language questions answering get failed. For the other questions such as *what/which, When, who*, our approach has good effect of Q&A.

4.4 RQ2: Inference Sub-Graph Effectiveness Evaluation

Inference Sub-Graph is a bridge between natural language and formal query statement. To better express the full semantics of a natural language question, during the generation of Inference Sub-Graph, we have to extend hidden nodes and hidden edges. For example, in the natural language question example *"Which issue written by Alex that modifies a method called by IndexWriter"* shown in Fig. 1, it is necessary to add three hidden nodes, i.e., *method, GitAuthor* and *Commit*, as well as two hidden edges, i.e., *have_method* and *code_mention*. In order to verify the role of the Inference Sub-Graph in the transformation from natural language to formal query, we statistically analyze the number of hidden node elements added to the Inference Sub-Graph. As shown in Fig. 3, 64% questions in Apache Lucene need to add hidden nodes in Inference Sub-Graphs, and as shown in Fig. 4, 55% questions in Apache Lucene need to add hidden edges in Inference Sub-Graphs. The same statistics are presented in Apache Nutch project. This shows that in the process of converting natural language questions into formal query statements, more than half of the natural language questions need to be extended. If we don't use the Inference Sub-Graph to extend the hidden nodes and edges, but simply combine natural language tokens and construct formal queries, then more than half of the natural language questions cannot be correctly answered. Therefore, Inference Sub-Graph proposed in our approach can effectively solve the transformation between natural language and Cypher query.

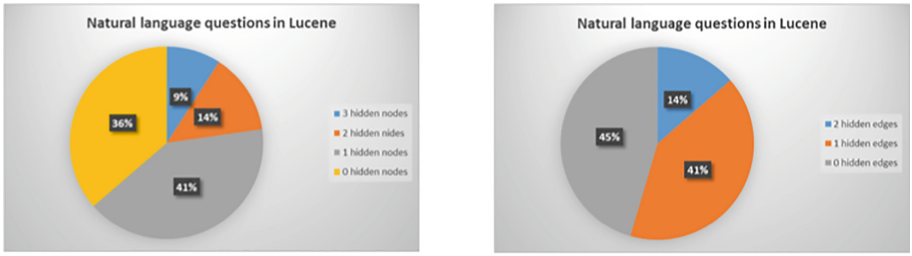

Fig. 3. The Distribution of Hidden nodes and edges in Inference Sub-Graph of Lucene Search Questions

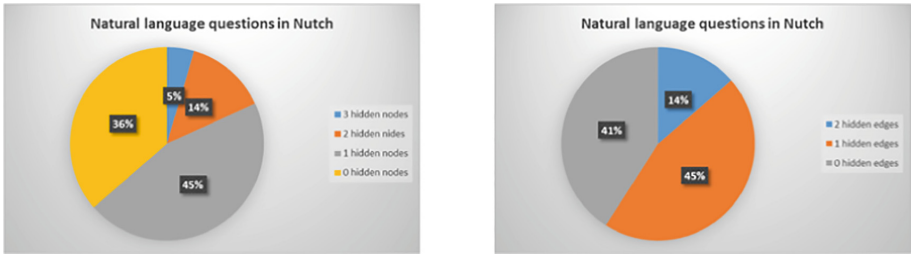

Fig. 4. The Distribution of Hidden nodes and edges in Inference Sub-Graph of Nutch Search Questions

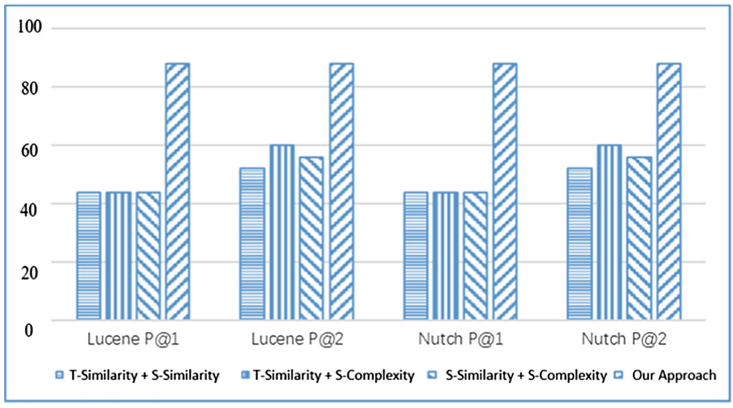

Fig. 5. The precise results of 3 kinds Measurement methods comparison analyse

4.5 RQ3: Measurement Strategy Comparison Evaluation

We designed a series of comparison experiments in order to compare the measurement effects of the three metrics: "textural similarity + structural similarity", "textural similarity + structural complexity", "structural similarity + structure complexity", so we can verify the role of each metric in the metrics. Figure 5

shows the experiment results: As shown in Fig. 5, It can be seen that, considering the correct Cypher in top-1, the accuracy of the two measurement features alone is about 50%, while the accuracy of our method (the combination of the three measurement features) is up to 94%. Considering the correct Cypher in top-2, the accuracy achieved by the measurement strategies of "textual similarity + structural similarity", "textual similarity + structural complexity" and "structural complexity + structural similarity" is 55%, 63% and 59% respectively, while the accuracy of our method can reach 97%. It means that if we discard any metric feature, the accuracy of the question answering will drop dramatically. This also proves that the three metrics we propose are reasonable.

5 Related Work

A lot of work have tried to construct a natural language query interface for graph database/knowledge [12], which mostly adopts NoSQL database. The existing researches mainly include: (1) approaches based on syntactic analysis; (2) approaches based on machine learning.

The basic process of the approach based on syntactic analysis [7] is as follows: firstly, the natural language query is parsed and its syntactic dependency tree is constructed, then the query transformation is carried out through methods such as node matching and rule extension, and finally formal query statement is obtained. Typically, Berant et al. [16] proposed to train a semantic parser, which is used to conduct natural language questions answering on the knowledge base. The basic process is: using the trained semantic parser to parse the input natural language questions into a logical form system, and then searching the answer from the knowledge base based on the structured logical expression; Based on the work of DCS-L, Yao et al. [17] located candidate entities in the knowledge base and constructed topic maps as candidate answers, and established feature vectors for the natural language questions and candidate answer entities, and converted the original problem into binary classification of candidate answers. Among the advantage of this method is based on formal query results, the user can determine the reliability of the query results. But in these research work, the words in the natural language query input by the user still need to be explicitly corresponding to a certain information (table name, attribute name, record, etc.) in the database table, otherwise the syntax tree is incomplete and the correct answer cannot be obtained.

In terms of the research based on machine learning method for the natural language query of knowledge base, the early work is mainly to record the feedback information when the user makes the query and use it to assist the next query according to the historical query information. Freitas et al. [11] utilize interactive algorithms to optimize query results, design user feedback methods to improve accuracy. The basic idea is to record feedback information when users input query, and then historical queries is used to assist the next query input. This method can optimize most methods, but it needs to be used more frequently to accumulate the user's usage history. Tunstall-Pedoe et al. [9] utilize human

factors in the process of generating templates. Zheng et al. [10] extract the correct natural language question template from the existing questions from Yahoo and other communities. Other recent representative work mainly rely on machine learning including sequence model, neural network model, attention mechanism, etc. [18,19]. Especially, the deep learning approach has become one of the main technologies of the knowledge base natural language question answering [21, 22]. Wen-tauYih et al. [23] defined a query sub-graph method, and the query sub-graph can be directly converted from the knowledge base to Logical formal system, the task of defining semantic parsing is to generate query sub-graph, use the knowledge base to perform pre-physical chain index, and use Deep CNN to calculate the matching degree of question and logical predicate, and obtain significant effect.

6 Conclusion

This paper proposes and implements a natural language question interface for software project knowledge graph. It extracts meta-model of software knowledge repository, constructs question related Inference Sub-Graph, then automatically transfers natural language question to structured Cypher query and returns the corresponding answer. We evaluated our approach with the open source software project Lucene and the open source software project Nutch's knowledge graph. The results show that our approach is effective in answering natural language questions from developers for software reuse. In the future, we will try to solve the more complex natural language question answering problems by increasing the human-computer interaction.

References

1. Fuqing, Y., Hong, M., Keqin, L.: Software reuse and software component technology. Acta Electronica Sinica **27**(2), 68–75 (1999)
2. Fuqing, Y.: Software reuse and its correlated techniques. Comput. Sci. **26**(5), 1–4 (1999)
3. Lin, Z.Q., Xie, B., Zou, Y.Z., et al.: Intelligent development environment and software knowledge graph. J. Comput. Sci. Technology **32**(2), 242–249 (2017)
4. McFetridge, P., Groeneboer, C.: Novel terms and cooperation in a natural language interface. In: Ramani, S., Chandrasekar, R., Anjaneyulu, K.S.R. (eds.) KBCS 1989. LNCS, vol. 444, pp. 331–340. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0018391
5. Lin, J., Liu, Y., Guo, J., et al.: TiQi: a natural language interface for querying software project data. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 973–977. IEEE Press (2017)
6. Mike, L., Mark, S.: A* CCG parsing with a supertag-factored model. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 990–1000, 25–29 October 2014
7. Li, F., Jagadish, H.V.: Understanding natural language queries over relational databases. ACM SIGMOD Rec. **45**(1), 6–13 (2016)

8. Tunstall-Pedoe, W.: True knowledge: open-domain question answering using structured knowledge and inference. AI Mag. **31**(3), 80–92 (2010)
9. Unger, C., Bühmann, L., Lehmann, J., et al.: Template-based question answering over RDF data. In: Proceedings of the 21st International Conference on World Wide Web, pp. 639–648. ACM (2012)
10. Zheng, W., Zou, L., Lian, X., et al.: How to build templates for RDF question/answering: an uncertain graph similarity join approach. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1809–1824. ACM (2015)
11. Freitas, A., de Faria F.F., O'Riain, S., et al.: Answering natural language queries over linked data graphs: a distributional semantics approach. In: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1107–1108. ACM (2013)
12. Krishna, S.: Introduction to Database and Knowledge-Base Systems, p. 18. World Scientific, Singapore (1992)
13. Siva, R., Mirella, L., Mark, S.: Large-scale semantic parsing without question answer pairs. Trans. Assoc. Comput. Linguist. **2**, 377–392 (2014)
14. Siva, R., Oscar, T., Michael, C., et al.: Transforming dependency structures to logical forms for semantic parsing. Trans. Assoc. Comput. Linguist. **4**, 127–140 (2016)
15. Zettlemoyer, L.S., Collins, M.: Online learning of relaxed CCG grammars for parsing to logical form. In: Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL), pp. 678–687 (2007)
16. Berant, J., Chou, A., Frostig, R., et al.: Semantic parsing on freebase from question-answer pairs. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 1533–1544 (2013)
17. Yao, X., Van Durme, B.: Information extraction over structured data: question answering with freebase. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, pp. 956–966 (2014)
18. Zhang, Y., Liu, K., He, S., et al.: Question Answering over Knowledge Base with Neural Attention Combining Global Knowledge Information. arXiv preprint arXiv:1606.00979 (2016)
19. Chen, J., Siva, R., Vijay, S., et al.: Learning structured natural language representations for semantic parsing. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 30 July–4 August, Vancouver, Canada, pp. 44–55 (2017)
20. Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G.: Learning semantic representations using convolutional neural networks for web search. In: Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, pp. 373–374 (2014)
21. Xiao, C., Marc D., Claire, G.: Symbolic priors for RNN-based semantic parsing. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017) (2017)
22. Yih, W.-T., Chang, M.-W., He, X., Gao, J.: Semantic parsing via staged query graph generation: question answering with knowledge base. In: Proceedings of the Joint Conference of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing of the AFNLP, vol. 1 (2015)
23. Chang, Z., Zou, L., Li, F.: Privacy preserving subgraph matching on large graphs in cloud. In: SIGMOD 2016, 26 June-01 July, San Francisco, CA, USA (2016)

24. Sun, Z., Wang, H., Wang, H., et al.: Efficient subgraph matching on billion node graphs. Proc. VLDB Endowment **5**(9), 788–799 (2012)
25. Zou, L., Huang, R., Wang, H., et al.: Answering, natural language question, over RDF a graph data driven approach. In: SIGMOD 2014, 22–27 June 2014, Snowbird, UT, USA (2014)
26. Li, W., Wang, J., Lin, Z., et al.: Software knowledge graph building method for open source project. J. Front. Comput. Sci. Technol. **11**(6), 851–862 (2017)
27. Li, H., et al.: Improving API caveats accessibility by mining API caveats knowledge graph. In: 2018 IEEE International Conference on Software Maintenance and Evolution ICSME (2018)