Cluster Comput (2019) 22:S847–S858
https://doi.org/10.1007/s10586-017-1332-3

Semantic query graph based SPARQL generation from natural language questions

Shengli Song[1] · **Wen Huang**[1] · **Yulong Sun**[1]

Received: 7 September 2017 / Revised: 25 October 2017 / Accepted: 3 November 2017 / Published online: 14 November 2017

Abstract In order to precisely represent natural language questions (NLQs) in question answering system (QAS) and provide a more naturally interactive mode, we require SPARQL, a formalized query patterns, instead of search expression to express the user's semantic query intention. However, how to generate and evaluate SPARQL query from NLQ is a mostly open research question. In this paper, we propose a framework that can help users translating NLQ into well-formed queries for knowledge based systems. We define a new graph structure, semantic query graph, and vocabulary to match all kinds of complex and compound questions without using domain ontology. Through query expansion and semantic query graph generation, the framework resembles subgraphs of the knowledge base and can be directly mapped to a logical form. Extensive experiments over NLQ in real RDF QASs verify the feasibility and efficiency of semantic query graph and our proposed framework with average F-measure of 0.825.

Keywords Query generation · Semantic query graph · SPARQL · Natural language question

1 Introduction

Recently, knowledge bases have attracted lots of attentions in both academia and industry. It is known that large-scale knowledge base like Freebase and DBpedia has been come to an important semantic database for supporting open domain question answering systems (QASs). It becomes necessary to interface SPARQL engines since it is impossible for an end-user to handle the complexity of the "schemata" of different pieces of knowledge bases. Therefore, the user requires having the capability of writing query sentences with a related query language, such as constructing SPARQL in Linked Data based QAS. In order to fill this gap, it is necessary to formalize natural language questions (NLQs) as SPARQL for capturing information in QAS directly. Also, we think that the availability of voice recognition services which understand natural speech and become more and more popular, especially on smartphones, implies that we have now to work on the translations of NLQ into formal queries.

Different from modern information retrieval systems which allow the user to locate documents that might contain the associated information, the QAS tries to retrieve accurate and concise answers. The wide variety of graph patterns that can be matched through SPARQL queries reflects the wide variety of the data that SPARQL is designed for the data of the Semantic Web. SPARQL can be used efficiently and effectively to extract the necessary information hidden in non-uniform data stored in various formats and sources. But the majority of QAS leaves it to the user to extract the useful information from an ordered list, such as the input question "What is the capital of China?" should get back response "Beijing" instead of users exploring a list of relevant documents presented to find an accurate answer.

With the development of QAS, it enables users to access the knowledge base by asking questions and getting back a proper response in concise words. There are many state-of-the-art methods to apply SPARQL to QAS. Through the introduction of SPARQL, it can more accurately express user's query intention so as to improve the accuracy of QAS, but SPARQL is so professional that it is hard to learn for most users. So, it is very important that how to generate SPARQL query automatically. Xser [13] is a semantic question sys-

✉ Shengli Song
xidiansls@163.com

[1] Software Engineering Institute, Xidian University, Xi'an, China

tem that proposes a question structure and a template which can generate SPARQL to query answer from KB. QAKiS [2] queries several multilingual versions of DBpedia at the same time by filling the produced SPARQL query with the corresponding language-dependent properties and classes. Thus, QAKiS can retrieve correct answers even in cases of missing information in the language-dependent knowledge base. SWIP [16] is a hybrid QAS that generates a pivot query, a structure between NLQ and the formal SPARQL target query.

However, the existing methods for translating NLQ into SPARQL query can cause a bit drop-out especially in relation mapping, which may influence the accuracy of query generation. Besides, these methods are almost for simple sentences but do not consider compound sentences or complex sentences.

In this paper, we propose a framework to generate SPARQL via constructing semantic query graph (SQG) based syntactic analysis and dependency parsing by considering the compound sentence and complex sentence. The generated SPARQL statement is executed over RDF graphs. Recently, there exists systems can complete this query such as Jena [17], RDF-3X [7], g-store [5].

The contributions of this paper are summarized as follows:

(1) We propose SQG to express semantic information and query intention that contained in complex and compound NLQ. Therefore, we can use the SQG to implement translations from all kinds of NLQ to SPARQL.
(2) Query expansion is used to promote completeness of the generated query. It is achieved by expanding synonyms or coordinative semantic entities of each original entity in questions. This expansion could be applied both in our proposed framework and existing ontology-based methods.
(3) The proposed SPARQL generation framework is not restricted by domain ontology, and it can be applied to more extensive areas. Triples and semantic information (entities and their relationships) are extracted directly from NLQ by analyzing their syntactic structures and expressed by SQG. SPARQL query could be generated by traversing the graph.
(4) The experimental results show that the proposed framework can successfully generate SPARQL queries in several data sets with average F-measure of 0.825.

The remainder of this paper is organized as follows. Section 2 introduces some definitions of SQG. The new proposed framework for translating NLQ into SPARQL is presented in Sect. 3. In the next section, we describe our experiments and results analysis in detail. We give some related works of state-of-the-art research about SPARQL generation in Sect. 5, which is followed in Sect. 6 by some conclusions and directions of future work.

2 Semantic query graph

2.1 SPARQL query structure

The definition of a formal semantics for SPARQL has played a key role in the standardization process of this query language. Although taken one by one the features of SPARQL are intuitive and simple to describe and understand, it turns out that the combination of them makes SPARQL into a complex language. A formalization of a semantic SPARQL is beneficial for several reasons, including to serve as a tool to identify and derive relations among the databases or KBs. These databases or KBs can stay hidden in the use case in order to identify redundant and contradicting notions, to drive and help the implementation of query engines, and to study the complexity, expressiveness. The power of SPARQL together with the flexibility of RDF can lead to lower development costs passing merging results from multiple data sources.

A SPARQL query consists of a set of triple patterns in which each element, i.e., subject, predicate, object, can be a variable. And SPARQL query has four types of queries: *SelectQuery*, *ContructQuery*, *DescribeQuery* and *AskQuery*. The four types of SPARQL query are defined by Definition 1–4.

Definition 1 (*SelectQuery*) The *SELECT* clause in a SelectQuery selects a group of variables, or all of them using as in SQL: the wildcard *. In this type of queries, one can eliminate duplicate solutions using DISTINCT.

SelectQuery= "SELECT" ("DISTINCT" | "REDUCED")? (Var+ | "")*
*"FROM" (DefaultGraphClause | NamedGraphClause) **
"WHERE"? OrderClause? LimitOffsetClauses?

Definition 2 (*ContructQuery*) In a ConstructQuery, the *CONSTRUCT* form, and more specifically the *ConstructTemplate* form, is used to constructs an RDF graph using the obtained solutions. The formalization of ConstructQuery is as follows:

ConstructQuery = " CONSTRUCT" ConstructTemplate
*"FROM" (DefaultGraphClause | NamedGraphClause) **
"WHERE"? OrderClause? LimitOffsetClauses?

Definition 3 (*DescribeQuery*) In a DescribeQuery, the *DESCRIBE* form is not normative. And it is intended to describe the specified variables or URIs or IRIs, i.e., it can return all the triples in the KB involving in these resources. The definition of DescribeQuery is as follows:

DescribeQuery = "DESCRIBE" (VarOrIRIref+ | "")*
*"FROM" (DefaultGraphClause | NamedGraphClause) **
"WHERE"? OrderClause? LimitOffsetClauses?

Definition 4 (*AskQuery*) In an AskQuery, the *ASK* form has no parameters but the answer can be queried. It returns TRUE

if the solution set is not empty, and FALSE otherwise. The AskQuery is described as follows:

AskQuery = "ASK"
*"FROM" (DefaultGraphClause | NamedGraphClause) **
"WHERE"?

The basic engine of the language is a pattern matching facility, which uses some graph pattern matching functionalities (sets of triples can also be viewed as graphs). From a syntactic point of view, the overall structure of SPARQL query language includes three main blocks:

- *SELECT* clause, which specifies the final form in which the results are returned to the user. SPARQL allows several forms of returning the data: a table using *SELECT*, a graph using *DESCRIBE* or *CONSTRUCT*, or a *TRUE/FALSE* answer using *ASK*. Among them, *ASK* whether there is at least one match of the query pattern in the RDF graph data; *CONSTRUCT* an RDF graph by substituting the variables in those matches in a set of triple templates; *DESCRIBE* the matches found by constructing a relevant RDF graph.
- FROM clause, which specifies the sources to be queried.
- WHERE clause, which is composed of a graph pattern. Informally speaking, this clause is given by a pattern that corresponds to an RDF graph where some resources have been replaced by variables. Not only that, more complex expressions (patterns) are also allowed, which are formed by using some algebraic operators. This pattern is used as a filter of the values of the dataset to be returned.

2.2 Semantic query graph definition

In order to organize large amounts of semantic information into one unified structure, we design a new representable graph structure, SQG, for SPARQL query transformation and generation. The SQG not only can express the semantic information of NLQ but also can map to SPARQL query language. The vertex and edge of SQG are defined according to the standard notions of RDF and SPARQL representation, i.e., triple, RDF graph, basic graph pattern. The variable vertex "?x0" can represent the variable in SELECT clause in SPARQL. Each relation triple in SQG can map to the graph pattern in WHERE clause. The specific explanation of SQG definition is as in Definition 5.

Definition 5 (*Semantic query graph*) SQG can be denoted as Q^s. $Q^s = (V, E)$, where $\boldsymbol{V}$ is a set of vertices that represent the named entity, variable or relation term in NLQ. The entity node is the named entity while variable node represents the common noun which stands for the category. $\boldsymbol{E}$ refers to a set of relational triples which is a set of relational phrases in NLQ.

In more details, $\mathbf{V}$ can be divided into three types: *answer node*, *entity node* and *variable node*. The three types of nodes and the edge of relation triple are described as follow:

- *answer node* (?x0) is used to map entities retrieved by the query statement.
- *entity node* is an existing entity in the knowledge base.
- *variable node* (?xi) implies some potential entities which are linked to the answer variable or entities which link to answer.

The relation triples of $\boldsymbol{E}$ are that relation edge connects two nodes in SQG. In natural language, the relation is usually a verb, and sometimes a noun or an adjective or a preposition. Corresponding to SPARQL statements, the WHERE clause of SPARQL query contains these semantic relation triple.

According to the three types of natural language sentences, such as simple sentence, complex sentence and compound sentence, NLQ can be deeply divided into three patterns [16]: (i) It is simplest, and it required single variable and single relation triple. i.e., "Who started pixar?". (ii) This pattern involves arithmetic operation (ranking or superlative). i.e., "Who did jackie robinson first play for?". It will generate single or multiple relation triples. (iii) The last pattern involves preposition phrase and the triple generated as the second case. i.e., "Who is the youngest player in the Premier League?". According to the structure of sentences, SQG can be represented as different patterns. In general situation, the NLQ is composed of a variable node and a relational phrase whose label is a verb. The more complex sentence is the sentence like "What is the mother of the husband of Jane?" which implies more than one variable node. The compound sentence and complex sentence may contain more than one relation which is tagged as ontology in knowledge base instead of property.

Given a question "Who created Goofy?", its SQG is as shown in Fig. 1. It contains a variable node which is an answer node, an entity node and a relation edge. The answer node ?x0 will be included in SELECT clause. After identifying the URIs of entities and relations in SQG, it needs to traverse each relation triple of SQG and generate SPARQL statements in order. The SPARQL elements are listed in Table 1. It can be found that relation triple $\langle ?\ x\ 0$, creator, Goofy$\rangle$ can be recognized and mapped to the triple pattern of RDF basic query. So the WHERE clause of question is "?x0 dbp:creator dbr:Goofy" that is in combination with information of knowledge base. In order to improve the accuracy of the answer, we add the types information of the answer based on the types

Fig. 1 The semantic query graph of a simple natural language question "Who created Goofy"

Table 1 Semantic query graph formulation of "Who created Goofy?"

	SQG elements	Mapping to SPARQL query
Answer node	?x0	SELECT ?x0
Entity node	Goofy	dbr:Goofy
Variable node	?x0	?x0 rdf:type rdo:Person
Relational triple	⟨? x 0, creator, Goofy⟩	WHERE {? x 0 rdf:type rdo:Person ?x0 dbp:creator dbr:Goofy}

of questions, such as Yes-No question, WH question, Tag question, Choice question, Hypothetical question, Embedded question and Leading question.

3 Transformation framework for SPAQRL generation

Based on the definition of SQG, we propose a new transformation framework for translating NLQ to SPARQL query. The overview framework is shown in Fig. 2. Precisely, NLQ could be converted into SPARQL statement through four main steps according to the framework: Query Dependency Parsing, SQG Construction, Entity and Relation Identification, and SPARQL Query Generation. In particular, an NLQ is dealt with the process of query dependency parsing to obtain its related dependency tree, which aims at revealing the syntactic structure of question by analyzing the relation between words. For example, the question "In which films directed by Garry Marshall was Julia Roberts starring?". After dependency parsing, the result is as follow: nsubj(starring, Julia Roberts), prep in(starring, film), prep in(film, Garry- Marshall), det(film, which), advmod(film, directed). When getting the parsing result of the question, we construct semantic query graph via an algorithm in Sect. 3.2. SQG can represent some triples:⟨Julia Roberts, starring, ?x0⟩,⟨?x0, type, film⟩ ⟨?x0, directed, Garry Marshall⟩. Finally, SQG is mapped to SPARQL query, which has completed translating NLQ to SPARQL query.

3.1 Question dependency parsing

The fundament of query dependency parsing is that the syntactic structure of NLQ is determined by a set of dependency relations and by a topic phrase and some dependency phrases. Query dependency parsing includes two main tasks. The first task is to analyze the structure of NLQ by constructing Dependency Parsing Tree (DPT), which can be completed by NLP parsing tools, such as the Stanford Parser. The second task is optimization process for DPT refinement so as to generate SQG.

Definition 6 (*Dependency parsing tree*) Dependency parsing tree can be denoted as DPT. DPT = (W, E, L), W = $W0, W1, \ldots, Wn$, W is the set of words of the question, Wi is a word of a sentence. E = $\langle Wh, Wm, l\rangle : 0 \leq h \leq n, 1 \leq m \leq n, l \in$ L, $\langle Wh, Wm, l\rangle$ indicates an edge from a head word (parent node) Wh to a modifier word (child node) Wm, l is the type of relationship between words. L is a set of the semantic relations of words. $W0$ is the head work of the whole NLQ. Figure 2 depicts a dependence structure of question "Which singer from London is married to David Beckham born in a city of London?", in which, the

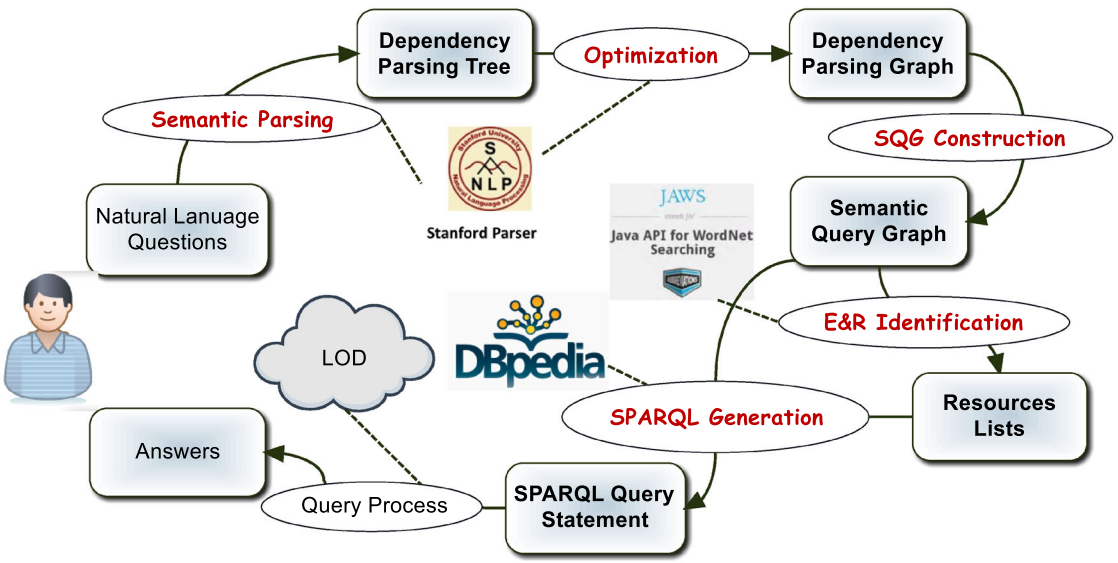

Fig. 2 Overview of the transformation framework for SPARQL generation

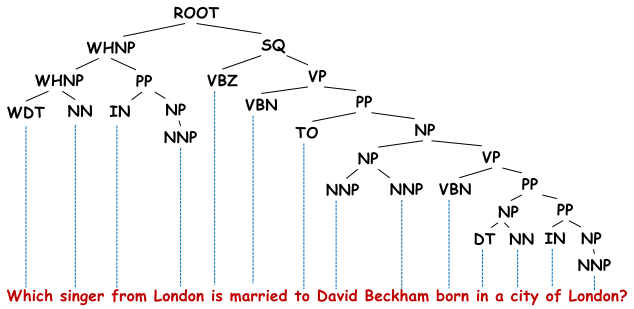

Fig. 3 The sample dependency parsing tree of the natural language question "Which singer from London is married to David Beckham born in a city of London?"

number represents the word position index in question. The root node "married" is the head word of the whole NLQ. The ⟨"married", "singer", "nsubj"⟩ represents that the word "singer" modifies "married" and the type of relation nominal subject (nsubj) (Fig. 3).

Optimizing operations should be applied to the DPT, and we will get an optimized Dependency Parsing Graph (PDG). The operations can be divided into three types. The first type is that merging the multi-words or compound expressions whose dependency is "n-n" (noun compound modifier, i.e. man-made) or "compound", such as the name of the person (i.e. David Beckham). The second type is to remove dependencies which are less semantic and are unlikely expressed in the triple pattern of knowledge base, such as "aux" (i.e. be, should), "determiner" (i.e. a, the), but if the dependent of "determiner" is a question word (i.e. what, which), the "determiner" should remain. The third type is joining dependencies which stand for conjunctive or disjunctive statements. Based on these three kinds of operations, the PDG of the example, "Which singer from London is married to David Beckham born in a city of London?", is shown as in Fig. 4. "David" and "Beckham" nodes are merged to one node "David Beckham". The dependency "auxpass" and the node "is" should be removed from DPT.

3.2 Semantic query graph construction

The SQG is constructed according to the given the dependency structure of NLQ statement. On the revision of dependence, the influence of prepositions or passive on subject and relation and the relationship between two associated entities are considered. So in this step, the dependency "proposition" should be reserved.

The detail of SQG construction algorithm is shown in Algorithm 1. "addEntity" is a function that constructs the entity node of SQG if the part of speech of Wi is "NNP"(i.e. named entity) . "createVarible" is used to construct the variable node. If the part of speech of Wi is "NN" or "NNS" (i.e. common noun "singer") and the id is the number of variables, we define ?x0 to be the answer node which presents the answer to SELECT statement. If Wi has an edge connecting the WH question (i.e. which, what) and it will also be denoted as ?x0. "createEdge" has three arguments that express the start node, the end node and the relation between them. The SQG of question "Which singer from London is married to David Beckham born in a city of London?" is shown in Fig. 6. From the figure, we add an edge between "David Beckham" and "?x0" which corresponds to "singer". The node of "born" is a verb so we add an edge between "David Beckham" and "?x1" which corresponds to "city". Because "singer" and "London" have already existed in SQG, we need to add an edge between "London" and "?x0" which corresponds to "singer", so as the nodes of "city" and "London".

Algorithm 1 Senmantic Query Graph Generation Algorithm

Input: $DependencyParsingTree$
Output: $SenmanticQueryGraph$
Set id is the nunber of variable node of SQG and id=0
for $W_i \in W$ **do**
  if W_i is "NNP" **then**
    $addEntity(W_i, V)$
  end if
  if W_i is "NN" or "NNS" **then**
    $createVarible(V, id)$ denoted as $?x_i$
    $addEntity(W_i, V)$
    $createEdge(?x_i, W_i, 'type')$
  end if
  if W_0 is verb **then**
    get the W_{msubj} of $< W_0, W_m, l >$ if l is "subj" or "nsubj" && W_m is noun
    get the W_{mobj} of $< W_0, W_m, l >$ if l is "obj" or "proposition" && W_m is noun
    $createEdge(W_{msubj}, W_{mob}, W_0)$
  end if
  if W_0 is "WHNP" **then**
    get $< W_0, W_m, l >$ if l is "subj" or "nsubj"
    if W_m is "NNP" **then**
      $createEdge(?x_0, W_m, 'name')$
    end if
  else
    if W_m is "NN" or"NNS" **then**
      if W_i is verb & $\neq 0$ **then**
        get the W_{hsubj} and W_{mobj} of $< W_{hsubj}, W_i, l >$, $< W_i, W_{mobj}, l >$
        $createEdge(W_{hsub}, W_{mobj}, W_i)$
      end if
      if $< W_h, W_m, l >, W_h, W_m \in V$ **then**
        $createEdge(W_h, W_m, l)$
      end if
    end if
  end if
end for

3.3 Entity and relation identification

When we get the SQG from the generation process, the entity nodes and the relation edges in SQG need to be connected with Linked Data in the knowledge base. The main idea of our method is to resolve the entity nodes to the resource recorded passing knowledge base and vocabulary.

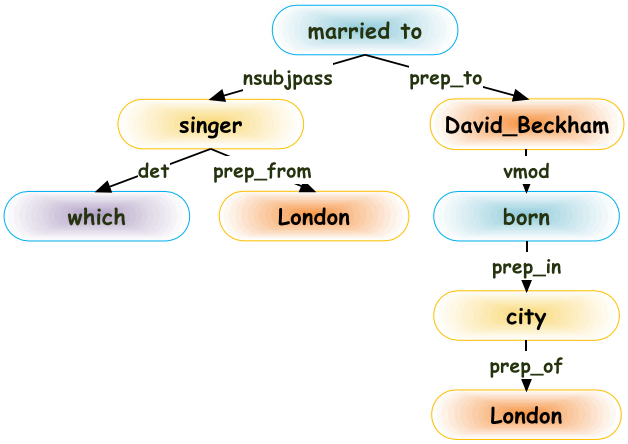

Fig. 4 The sample dependency parsing graph of the natural language question "Which singer from London is married to David Beckham born in a city of London?"

The vocabulary, which is extracted from special knowledge, is to mainly record the mapping between attributes and relationships commonly used in the knowledge base and natural language terms. The result is tagged as the resource, class, ontology or category. And the entity nodes do not need to pretreat before mapping because the form of representation of entity nodes are almost the same as those of resources in the knowledge base. i.e., the entity node "river" corresponds to the resource "⟨http://dbpedia.org/ontology/River⟩" in the knowledge base. However, the relation edges are not the case. It is more difficult to deal with than the entity nodes. The predicate can be classified into three categories including verbal predicate, adjectival predicate and predictive. The latter two categories can be processed together.

Among knowledge base (i.e., DBpedia, Yago), the semantic similarity of the object properties can be measured by using a string similarity score [26]. The score is calculated by the length of the greatest common subsequence over the length of the word. For example, the predicate "written" is searched for corresponding properties in a property list that have the greatest common subsequence. The object property dbont:writer is the most similar object property for the predicate "written". A list of all possible pairs of object properties is constructed from knowledge base with similar meanings. For each item, we have calculated the similarity score by using WordNet. By doing so, we get a list of object properties with their similar meanings. For example, dbont:writer has similar meaning with dbont:author.

If a predicate with POS tag value is noun or adjective, it searched for candidate data properties using the score of string similarity. For the question "What is the height of Michael Jordan?", the resulting triple will be $\langle Michael\ Jordan, height, ?x\rangle$. Using common subsequence of the predicate, "height" is mapped to dbont:height. A list of adjectives is constructed for all data properties defined by special knowledge base using Java API for WordNet Searching (JAWS). By doing so, we get a list of data properties with their adjective meanings. For the question "How tall is Michael Jordan?", the resulting triple will be: $\langle Michael\ Jordan, tall, ?x\rangle$. By using an adjective list obtained, the predicate "tall" is mapped to dbont:height.

QAS needs patterns that denote relations between entities. There would be cases where the intended NLQ can be phrased in numerous ways. For instance, a question about the birthplace of Michael Jackson can be built as "Where was Michael Jackson born in?" or "Where was Michael Jackson born?". However, in order to query data from DBpedia, we will need the object property of "birthPlace". For this reason, we have to construct a "birthPlace" relationship with "Michael Jackson" and "Gary, Indiana", where he was born in, for both questions. Here "born in" and "born" are phrases in natural language, so it is necessary to map such phrases into the same object property. Nakashole et al. [14] proposes a method that constructs a large resource for word or phrases patterns that denotes binary relations between entities, organizing patterns into a set of synonymous patterns similar in spirit to WordNet. The patterns are semantically typed and organized into a subsumption taxonomy called PATTY. Extracting word or phrases patterns from a corpus follows two steps: firstly, relational patterns from a corpus are compiled and then a semantically typed structure is imposed on them. After extracting the word or phrases patterns from a corpus, this work arranges the patterns in a semantic taxonomy. A prefix-tree for frequent patterns is used to determine inclusion, mutual inclusion, or independence. The prefix-tree stores support sets of patterns.

There are some common nouns standing for entity nodes, so it should be processed by using a different method [27–30]. When constructing SQG, we add a relation edge whose name is "type" between a variable node and an entity node. For a relation term t_i in a relationship set T, if t_i contains rdf:type predicate, and then the object of t_i is regarded as entity class of the knowledge base. For the question "Which book is written by Orhan Pamuk?" the word "book" is mapped to dbont:Book using label properties of all entity classes.

We map the relation edges to predicates and map the entity nodes to resources. It can get some candidate resource lists and predicate lists according to the former two steps. The next step is to recognize the correct relationship chain in candidate lists. In the sample query as shown in Fig. 4, "Which singer from London is married to David Beckham born in a city of London?" First, the entity node "David Beckham" is mapped to some URIs representing the David Beckham in DBpedia. The list includes "http://dbpedia.org/resource/DavidBeckham" and "http://dbpedia.org/resource/Category:DavidBeckham". The two entity nodes with common nouns are "London" and "singer". "London" is mapped to some URIs which are "http://dbpedia.org/Class/London"

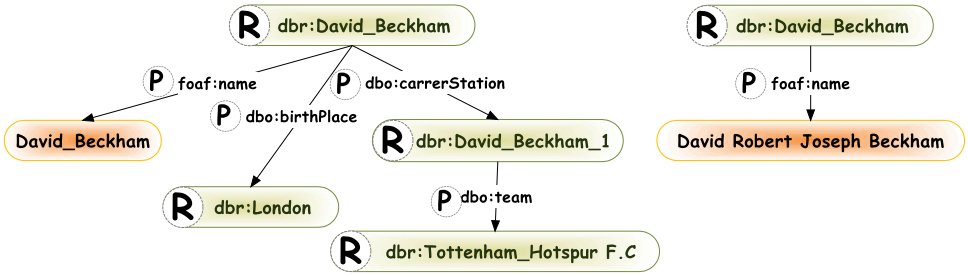

Fig. 5 Triple patterns of the resource "David Beckham"

and "http://dbpedia.org/resource/Category:London". The representation of "singer" is "http://dbpedia.org//London" and "http://dbpedia.org/resource/Category:London". When completing the mapping of resource and property, we need to denote the relationship between entities based on the result of the previous step.

3.4 SPARQL query generation

The new query presentation model, which has been built during the SQG construction step, is used to generate the SPARQL query statement. Given an SQG, we can translate it into SPARQL via traversing the whole graph. Each edge of SQG can be represented as one SPARQL condition statement. When we construct SQG and identify entity and relation based on the underlying knowledge base, we can traverse relation triple and generate $\langle subject, relation, object \rangle$ triple in order. The variable node stands for a variable in SPARQL and entity node can be seen as resource or attribute value. The directed edge presents the relationship between the node and will be used to generate predicate of SPARQL query. In this step, each SQG will be translated to SPARQL query based on the definition of SQG and SPARQL query language in Sect. 2.

The SPARQL query generation process should consider two different factors: (1) The generated SPARQL form which takes into account the expected type of the result form (e.g., ASK or SELECT); (2) The presence of aggregation operators and the variable associated with the question topic. In most cases, the generated SPARQL form is SELECT. If an aggregation operator is expected, it requires the GROUP BY clause which will be processed during the other process, that is the generation of WHERE clause. This part consists of the generation of strings for representing each relation triples for SQG and the filters if the predicates are negated terms. When aggregation operators are used, it is also necessary to recursively generate filters and sub-queries for computing the subsets of expressions before their aggregation. The result according to the intention of the user always has a single column.

An example is shown in Fig. 5. According to the sample SQG, the relation triples can be obtained (i.e., (*?x0, rdf:type, singer*), (*?x0, birthplace, London*), (*singer, marriedTo, David_Beckham*), (*David_Beckham, birthplace, ?x1*), (*London,cities_in_London,?x1*)). The resulting form, which can be determined based on the type of question, is SELECT. For WHERE clause, each triple can correspond with each query statements. So, we can generate the SPARQL statement of the sample question as shown in Fig. 6.

In Fig. 6, we merge the relational triples obtained from Fig. 5. Every triple can generate a graph including two nodes and one edge. We use "(David_Beckham, birthplace, ?x1)" as an example. There are two nodes named "David_Beckham" and "?x1", and the edge named "birthplace". The generated graph is as shown in the upper portion of Fig. 6. The graph can be transformed to the SPARQL query shown in the lower portion of Fig. 6 easily.

4 Experimentation

4.1 Question datasets

For the evaluation of our framework, we use four different datasets. The first three datasets are QALD [24], WebQuestions [1] and Free917 [3]. The QALD dataset consists of 149 questions annotated with the corresponding SPARQL query and answers over Linked Data. The WebQuestions dataset contains 5,810 question-answer pairs, collected using the Google suggest API and crowdsourcing. The Free917 dataset contains 917 questions annotated with logical forms grounded to Freebase. The fourth dataset is a new NLQ collections named "WebNLQ" which we collected via different kinds of questions and answers from web pages. The WebNLQ dataset comprises 215 questions, and most of them are complex and compound questions. We also make use of DBpedia and Freebase knowledge base for the query. The former contains 10 million entities, 1.8 billion relations between entities and several billions of triples covering sorts of the topics (i.e., people, geography, films, book, etc.). In Freebase, the subjects are called topics and the data is stored about depended on their type defined as predicates.

As shown in Table 2, we analyze the complexity of NLQ and classify them into three types according to NLQ structure: simple sentence (simplest, arithmetic, preposition),

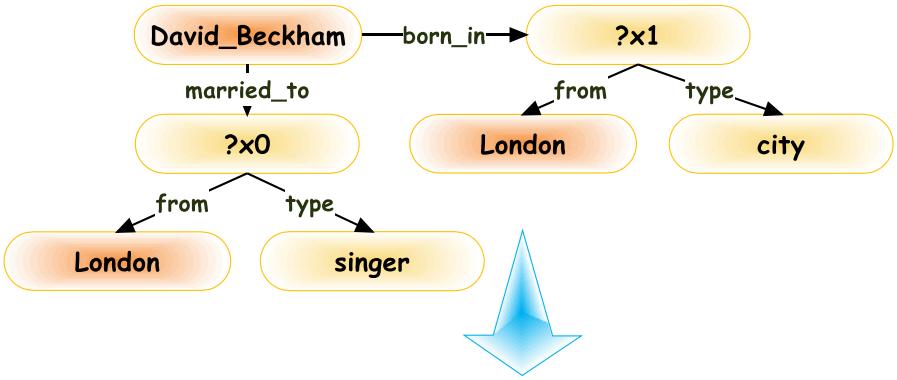

Fig. 6 The semantic query graph and generated SPARQL query of the natural language question "Which singer from London is married to David Beckham born in a city of London?"

```
SELECT ?x0 WHERE
{
    ?x0 rdf:type dbo:singer
    ?x0 dbp:birthPlace db:London
    ?x0 dbp:marriedTo dbr:David_Beckham
    dbr:David_Beckham dbp:birthPlace ?x1
    dbr:London Cities_in_London ?x1
}
```

Table 2 The classification of QALD

Type of question	Example	Number
Simple sentence	Simplest: Who produces TV?	43
	Arithmeticz: Who was the first to climb Mount Everest?	17
	Preposition: How many languages are spoken in Colombia?	72
Complex sentence (almost attributive clause)	What states are border states which the Mississippi runs through?	10
Compound sentence	What jobs desire a degree but don't use C++?	7

compound sentence and complex sentence. Then the questions can be classified into four types according to the type of answer: definition, affirmation, list, yes/no according to the logic and the type of answer. In addition to processing the question using question classification, more information about the expected answer of question can be derived (i.e., human is corresponding to the who question, date or time is corresponding to the when question, etc.).

4.2 Experimental setup

We first parse the question into words using Stanford Parser, and next find the synonyms terms for each word using WordNet. And we extract related terms for the query statement from knowledge base using DBpedia Spotlight. Synonyms terms and related terms are combined using permutation in WordNet, and we can compute semantic similarity by WordNet. For example "What is the name of football clubs in EEFA?", we use Stanford CoreNLP to syntactic analysis. The result is DPT structure about the POS tag of each word. We also use WordNet to analysis the synonyms form and find the synonyms terms form of noun phrases. So the expanded query of sample question is "what is the name of the football:soccer clubs in EEFA?"

We use SQG construction algorithm to construct the SQG for each sentence. There exist differences between natural language and knowledge in representation. In order to improve the accuracy of generated SPARQL statements, we build the vocabulary to record the commonly used predicate list of in DBpedia in to improve the accelerate linking speed. This vocabulary can be used to map the natural language to knowledge. Besides in this work, we use PATTY to extract relationship patterns.

As building WHERE clause blocks, we retrieve all relation triples from the SQG with the target entity as subject or object by issuing the queries t ?p ?o and ?s ?p t. Some of these triples need to be filtered out because they would spoil the query answer immediately. We filter out all triples which have non-stopword tokens in common with the target entity. For each semantic relation, three or more than three triple patterns are created. Using the SPARQL query generation procedures in Sect. 3.4, the SQG can be converted into SPARQL query statements.

4.3 Evaluation and analysis

We evaluate our experiment results with estimating answers return from generated SPARQL query. For the generated SPARQL results, we compute three metrics including precision (P), recall (R), and F-measure (F). The precision for each dataset is the number of true positive (SPARQL generated correctly) divided by the total number of positive (SPARQL generated) in the dataset. The recall for each dataset is the number of true positive divided by the total number of true (SPARQL should generated correctly) in the dataset. F-Measure for each dataset can be calculated by $2 \times \text{precision} \times \text{recall}/(\text{precision} + \text{recall})$. For QALD evaluation, we sort out the answer in QALD and manually create gold standards.

As shown in Tables 3, 4, and 5, the results of translating NLQ into SPARQL query obtained from different methods on different datasets. The methods include our proposed framework named "SQGTrans" and other various state-of-the-art translation methods, such as AskHow@QALD, QuerioDaLI@QALD, Xser@QALD, ONLI@WebQuestions and QuerioDaLI@ WebQuestions,

From the results of our proposed transformation framework "SQGTrans" in three open datasets, which are QALD, WebQuestions and Free917, the accuracy is higher than that of the other methods. The F1 value on the QALD benchmark of SQGTrans is 0.83, which is 15% higher than the state-of-the-art method Xser, the value of which is 0.72. The F1 value on the dataset WebQuestions is 0.83, which is the same as the state-of-the-art method ONLI. SQGTrans also obtains a higher F1 value on dataset Free917 than the state-of-the-art method QuerioDALI, and the values are 0.78 and 0.72 respectively.

Through a further study on the experimental results, there are 9.8% questions failed to generate SPARQL query, and 6.7% questions classified out of scope for DBpedia. There are more than 80% questions can be correctly translated and return correct answers.

The SQGTrans framework we proposed first conclude the relation of words via question classification and syntactic analysis, and then generate SQG according to the semantic information of NLQ. It traverses the whole SQG by starting from the variable node "?x0" instead of mapping to SPARQL template. We can accurately generate equivalent SQG of simple sentences. The result of translation from NLQ into SPARQL is divided into three types: incorrect query, correct query with a right answer, correct query with wrong answer. The failure analysis of the experiment results are as follows:

(1) SQG construction: Some questions are not processed successfully due to incorrect dependency analysis and incorrectly named entity recognition.
(2) Entity and relation mapping: In QALD dataset, there are 17 questions which could not map to DBpedia. Because some resources are in the form of noun while these words are adjective form, such as representation in question is "Germany cities" but there is not "Germany" in the knowledge base. These kinds of questions can't be translated to SPARQL query. And also in some cases, it could not resolve the relation to the correct DBpedia property. Besides, the result of relation mapping has much more connections with PATTY. PATTY taxonomy is restricted to a number of relational patterns because these corpora include a certain number of textual patterns that denote a relation between entity pairs. It does not make use of a universal corpus that can include nearly all available facts about entities or phrases combining entity pairs. PATTY includes some noisy data that is not related to the actual pattern; conversely, it harnesses patterns that have an opposite meaning against the actual pattern.
(3) DPT optimization: WordNet is frequently used but it suffers from two main problems : there is a lack of proper nouns which we tackle by using the Linked Data including DBpedia which contains mainly instances. A large number of ambiguous terms leads to a disambiguation problem. However, this does not manifest itself in the relatively small models of the benchmarks.

Table 3 Results on the QALD benchmark

	Recall	Precision	F-measure
AskHow	0.63	0.60	0.61
QuerioDaLI	0.69	0.64	0.61
Xser	0.71	0.72	0.72
SQGTrans	**0.79**	**0.87**	**0.83**

The number of bold is the maximum value in each column

Table 4 Results on WebQuestions and Free917

		Recall	Precision	F-measure
WebQuestions	ONLI	**0.85**	0.80	0.83
	SQGTrans	0.84	**0.82**	**0.83**
Free917	QuerioDALI	0.72	0.72	0.72
	SQGTrans	**0.75**	**0.82**	**0.78**

The number of bold is the maximum value in each column

Table 5 Results on WebNLQ

	Recall	Precision	F-measure
SQGTrans	**0.80**	**0.94**	**0.86**

The number of bold is the maximum value in each column

5 Related works

The SPARQL generation framework is very related to semantic QAS. In the following, we present existing QAS, which is launched on Linked Data so far. Xser is based on the observation, and it contains two independent steps. First, Xser determines the NLQ structure solely based on a phrase level dependency graph. The next stage is using the target knowledge base to instantiate the predefined template. However, when moving to another domain based on a different knowledge base, it only affects parts of the approach so that the conversion effort is lessened. QAKiS queries several multilingual versions of DBpedia at the same time by filling the produced SPARQL query with the corresponding language-dependent properties and classes. Thus, QAKiS can retrieve correct answers even in cases of missing information in the language-dependent knowledge base. SWIP generates a pivot query, a hybrid structure between the natural language question and the formal SPARQL target query. Generating the pivot queries consists of three main steps: (1) named entity identification; (2) query concept identification; (3) subquery generation based on the rules of query construction. To formalize the pivot queries, the query is mapped to linguistic patterns, which are created by hand from domain experts. If there are multiple applicable linguistic patterns for a pivot query, the user chooses between them. And there are some state-of-art methods of translating NLQ into SPARQL. The three kinds of approaches which are in different aspects are introduced as follows.

5.1 Ontology-based translation

Lehmann et al. [9], Paredes-Valverde et al. [15], Sander et al. [18] and Tatu et al. [23] firstly convert user input question into query tree which represents the gap between natural language and SPARQL. And then they combine domain specific knowledge into ontologies, lexicons, and rules for creation of SPARQL query. Song et al. [20,21] mainly translate NLQ to logic intermediate language via a feature-based grammar with semantic and propose an auto-suggest mechanism steers NLQ which are generated answers from the knowledge base. Matteis et al. [12] describes Treo, a natural language query mechanism for Linked Data. Treo firstly determines the key entities presentation of question which is the input of query parsing, and create a logic form, partially ordered dependency structure after parsing. Treo provides an approach which combines PODS and spreading activation search for handling semantic match between NLQ and Linked Data.

5.2 SPARQL generation using template

Horridge et al. [8] and Shekarpour et al. [19] propose a frame- work SINA that extracts input question into keywords and map keywords to a set of entity identifiers to cope with the resource disambiguation. After that, it defines query graph pattern as SPARQL template according to the standard notions of the RDF and SPARQL specification. Lopez et al. [10], Marginean [11], Pradel et al. [16] and Zheng et al. [26] define the SPARQL interpretation process consists named entities, dependency parse, construct a new query according to dependency parse tree and map the new query into predefined query patterns.

5.3 Semantic parsing based translation

Ferré [6], Tatu et al. [22], Xu et al. [25] and Dubey et al. [4] summarize a phrasal semantic parsing based SPARQL generation method. Firstly, they detect phrases segmented from question via defining four types of words (i.e., variable, category, entity, and relation). And then, it parses question to analysis relations between words. Finally, it combines the result of phrases detecting and dependency parsing to build a model directed acyclic graph and translate it to SPARQL query via underlying knowledge.

6 Conclusions

In the QAS based on Linked Data, it is a crucial task that the NLQ should be parsed to SPARQL statements automatically. In this paper, we proposed a transformation framework for generating formal SPARQL query from NLQ, especially a method for constructing SQG that can greatly improve the conversion efficiency and accuracy. Until now, the method is the excellent one, in contrast to the state-of-the-art methods in unsupervised filed. The new framework also considers the dependency parsing optimization, recovery the sentence structure and coreference resolution to complete the semantic information of NLQ. Currently, our method uses Stanford Parser to get the optimizing DPT and construct SQG. And secondly, in the stage of mapping resource, candidate terms sometimes are improper, and we should provide an online selection method of the candidate words as the future work, which can greatly reduce the mapping time, and improve the overall performance.

References

1. Berant, J., Chou, A., Frostig, R., et al.: Semantic parsing on freebase from question-answer pairs. EMNLP **2**(5), 6 (2013)
2. Cabrio, E., Cojan, J., Gandon, F, et al.: Querying multilingual DBpedia with QAKiS. In: Extended Semantic Web Conference. Springer, Berlin, pp. 194–198 (2013)
3. Cai, Q., Yates, A.: Large-scale semantic parsing via schema matching and lexicon extension. ACL **1**, 423–433 (2003)

4. Dubey, M., Dasgupta, S., Sharma, A., et al.: Asknow: a framework for natural language query formalization in SPARQL. In: International Semantic Web Conference, pp. 300–316. Springer, Cham (2016)
5. Faleiro, J.M., Abadi, D.J.: FIT: A Distributed Database Performance Tradeoff. IEEE Data Eng. Bull. **38**(1), 10–17 (2015)
6. Ferré, S.: Sparklis: an expressive query builder for SPARQL endpoints with guidance in natural language. Semant. Web **8**(3), 405–418 (2017)
7. Hasan, A., Hammoud, M., Nouri, R., et al.: DREAM in action: a distributed and adaptive RDF system on the cloud. In: Proceedings of the 25th International Conference Companion on World Wide Web. International World Wide Web Conferences Steering Committee, pp. 191–194 (2016)
8. Horridge, M., Musen, M.: Snap-SPARQL: a java framework for working with SPARQL and OWL. In: International Experiences and Directions Workshop on OWL, pp. 154–165. Springer, Cham (2015)
9. Lehmann, J., Bühmann, L.: Autosparql: let users query your knowledge base. Semanti. Web Res. Appl. 63–79 (2011)
10. Lopez, V., Tommasi, P., Kotoulas, S., et al.: Queriodali: question answering over dynamic and linked knowledge graphs. In: International Semantic Web Conference, pp. 363–382. Springer, Berlin (2016)
11. Marginean, A.: Question answering over biomedical linked data with grammatical framework. Semant. Web **8**(4), 565–580 (2017)
12. Matteis, L., Hogan, A., Navigli, R.: Keyword-based navigation and search over the linked data web.In: LDOW@ WWW (2015)
13. Mazzeo, G.M., Zaniolo, C.: Answering controlled natural language questions on RDF knowledge bases. In: EDBT, pp. 608–611 (2016)
14. Nakashole, N., Weikum, G., Suchanek, F.: PATTY: a taxonomy of relational patterns with semantic types. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 1135–1145 . Association for Computational Linguistics (2012)
15. Paredes-Valverde, M.A., Rodríguez-García, M.Á., Ruiz-Martínez, A., et al.: ONLI: an ontology-based system for querying DBpedia using natural language paradigm. Expert Syst. Appl. **42**(12), 5163–5176 (2015)
16. Pradel, C., , Haemmerlé, O., Hernandez, N.: Natural language query interpretation into sparql using patterns. In: Proceedings of the Fourth International Conference on Consuming Linked Data, vol. 1034, pp. 13–24. CEUR-WS.org (2013)
17. Qiang, L.: Painting semantic retrieval algorithm research based on ontology. In: Proceedings of the 2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA), pp. 621–624. IEEE (2015)
18. Sander, M., Waltinger, U., Roshchin, M., Runkler, T.: Ontology-based translation of natural language queries to SPARQL. In: Proceedings of the Natural Language Access to Big Data. AAAI 2014 Fall Symposium. Citeseer (2014)
19. Shekarpour, S., Marx, E., Ngonga Ngomo, A.-C., Soren, A.: Semantic Interpretation Of User Queries For Question Answering On Interlinked Data. Elsevier-Web Semantics (2015)
20. Song, D., Schilder, F., Smiley, C., et al.: Natural language question answering and analytics for diverse and interlinked datasets. In: HLT-NAACL, pp. 101–105 (2015)
21. Song, D., Schilder, F., Smiley, C., et al.: TR discover: a natural language interface for querying and analyzing interlinked datasets. In: International Semantic Web Conference, pp. 21–37. Springer, Cham (2015)
22. Tatu, M., Balakrishna, M., Werner, S., et al.: Automatic extraction of actionable knowledge. In: Proceedings of the 2016 IEEE Tenth International Conference on Semantic Computing (ICSC), pp. 396–399. IEEE (2016)
23. Tatu, M., Balakrishna, M., Werner, S., et al.: A semantic question answering framework for large data sets. Open J. Semant. Web (OJSW) **3**(1), 16–31 (2016)
24. Unger, C., Forascu, C., Lopez, V., et al.: Question answering over linked data (QALD-4). Working Notes for CLEF 2014 Conference (2014)
25. Xu, K., Zhang, S., Feng, Y., et al.: Answering natural language questions via phrasal semantic parsing. In: Natural Language Processing and Chinese Computing, pp. 333–344. Springer, Berlin (2014)
26. 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)
27. Zhou, Q., Luo, J.: The study on evaluation method of urban network security in the big data era. Intell. Autom. Soft Comput. (2017). https://doi.org/10.1080/10798587.2016.1267444
28. Zhou, Q.: Multi-layer affective computing model based on emotional psychology. Electron. Commer. Res. (2017). https://doi.org/10.1007/s10660-017-9265-8
29. Zhou, Q.: Research on heterogeneous data integration model of group enterprise based on cluster computing. Clust. Comput. **19**, 1275–1282 (2016). https://doi.org/10.1007/s10586-016-0580-y
30. Zhou, Q., Luo, J.: The service quality evaluation of ecologic economy systems using simulation computing. Comput. Syst. Sci. Eng. **31**(6), 453–460 (2016)

Shengli Song is currently an associate professor with Xidian University, China. He received his Ph.D. degree in Computer Science and Technology from Xidian University, Xi'an, China, in 2011. His current research interests include semantic computing, text analytics and natural language processing.

Wen Huang is currently a master student majoring in Software Engineering in Xidian University. Her research interest is semantic data retrieval.

Yulong Sun is currently a master student majoring in Software Engineering in Xidian University. His research interest is mobile data management.