future internet

MDPI

Article

Complex Queries for Querying Linked Data

Hasna Boumechaal * and Zizette Boufaida

LIRE Laboratory, University of Constantine 2-Abdelhamid Mehri, Constantine 25000, Algeria
* Correspondence: hasna.boumechaal@univ-constantine2.dz

Abstract: Querying Linked Data is one of the most important issues for the semantic web community today because it requires the user to understand the structure and vocabularies used in various data sources. Furthermore, users must be familiar with the syntax of query languages, such as SPARQL. However, because users are accustomed to natural language-based search, novice users may find it challenging to use these features. As a result, new approaches for querying Linked Data sources on the web with NL queries must be defined. In this paper, we propose a novel system for converting natural language queries into SPARQL queries to query linked and heterogeneous semantic data on the web. While most existing methods have focused on simple queries and have ignored complex queries, the method described in this work aims to handle various types of NL queries, particularly complex queries containing negation, numbers, superlatives, and comparative adjectives. Three complementary strategies are used in this context: (1) identifying the semantic relations between query terms in order to understand the user's needs; (2) mapping the NL terms to semantic entities; and (3) constructing the query's valid triples based on the different links used to describe the identified entities in order to generate correct SPARQL queries. The empirical evaluations show that the proposed system is effective.

Keywords: semantic web; Linked Data; question answering systems; SPARQL

Citation: Boumechaal, H.; Boufaida, Z. Complex Queries for Querying Linked Data. *Future Internet* **2023**, *15*, 106. https://doi.org/10.3390/fi15030106

Academic Editor: Paolo Bellavista

Received: 21 December 2022
Revised: 17 February 2023
Accepted: 26 February 2023
Published: 9 March 2023

1. Introduction

On the web, heterogeneous, complex, and distributed datasets have emerged in recent years. Linked Data [1] refers to a collection of interconnected datasets on the semantic web. This last method uses ontologies to represent the semantics and structures of datasets. This massive amount of data available on the web must be in a resource description framework (RDF) data model to create a Web of Data [2]. An RDF is a World Wide Web Consortium (W3C) standard for describing and defining various types of Linked Data vocabularies.

In this context, one of the most pressing issues for the semantic web community today is querying Linked Data. However, the user must understand the structure and vocabulary used in various data sources. Furthermore, users must be familiar with the syntax of formal query languages such as SPARQL [3]. Because they are accustomed to natural language-based search, novice users may find it challenging to use these features. While forming SPARQL queries to query Linked Data is not the easiest task, even for advanced users, natural language interfaces are considered the ultimate solution of semantic web research for accepting natural language queries and translating them to SPARQL queries.

Over the years, various approaches for querying Linked Data with NL queries have been presented in the literature, such as QASYO [4], FREyA [5], DEANNA [6], GFMed [7], CASIA [8], gAnswer [9], Xser [10], WDAqua [11], the system in [12], and finally, the method by Kotnis et al. [13]. However, most existing methods have been designed to answer simple queries and do not take into account complex queries with specific conditions. They are not intended to answer compound queries or queries containing negation, numbers, date comparisons, superlatives, and comparative adjectives.

This work primarily aims to propose a Linked Data query framework for answering various NL queries. Simple queries, as well as those containing conjunctions, negations,

numbers, superlatives, and comparative adjectives, can be addressed by our proposed system. A new system for constructing correct SPARQL queries from various NL query scenarios, particularly complex queries, is proposed. The following steps are used to interpret the user's query:

1. Identify the semantic relations between the NL query terms;
2. Map these terms to specific entities of Linked Data;
3. Combine the identified semantic relations, the matching entities, and the links between these matching entities in the datasets to form valid triples of the query;
4. Formulate SPARQL queries based on all the valid triples.

The remainder of this paper is organized as follows: Section 2 presents existing works in the field of Linked Data querying; Section 3 discusses the proposed system and its main steps; then, an evaluation is presented in Section 4; Section 5 conducts a discussion; and Section 6 presents the conclusions and perspectives of our work.

2. Related Work

Various approaches in the literature have been used to analyze a user's natural language query to query Linked Data. This section provides information about some approaches used on the semantic web:

Triple-based system approaches. These systems mainly focus on natural language processing (NLP) tools to represent the syntax tree of the NL query as a set of triples in the form <subject, predicate, object>. Then, these query triples are translated into dataset compatible triples to produce a structured query. Specifically, they use NLP tools, semantic metrics, and WordNet [14] to make sense of NL queries with respect to a dataset's structure. QASYO [4] is an example of these systems. These systems require syntactically and grammatically correct queries as input in this context. However, once the query is parsed incorrectly or contains syntax errors, producing the correct structured query and the relevant answers is difficult. This problem becomes worse, especially for complex queries.

Graph-based system approaches. These systems represent the structure of the query as a graph based on the dependency analysis of the query. Then, this graph is mapped to a subgraph in the RDF dataset. In particular, they search in the RDF graph for the semantic elements that correspond to the query terms to construct the semantic dependency graph, which can be easily converted into a structured query. DEANNA [6], gAnswer [9], WDAqua [11], the system in [12], and the method by Kotnis et al. [13] can be classified as graph-based systems. However, identifying the corresponding entities between the graphical representation of the query and the RDF graph is not a trivial task because these systems must measure the similarity between the two graphs to find the RDF subgraphs corresponding to the query. Therefore, these systems need to summarize the knowledge graph to reduce the search space. It should be noted that these system approaches do not address complex queries because identifying the graphical representation of complex queries and their corresponding entities with the RDF graph is very difficult compared to simple queries.

Machine learning-based system approaches. These systems use a contextual learning mechanism to improve system performance over time by answering specific user questions. FREyA [5] and CASIA [8] are systems that use machine learning to record suggestions presented to the user for the disambiguation of specific terms. These systems are explainable for a particular dataset. However, the main limitation of these system approaches is their inability to consider complex queries, and they have difficulty answering queries for which they have not seen training data.

Controlled language-based system approaches. In this type of system, users express their input queries in the syntax of a controlled language (CL). A CL is a restricted version of a natural language that uses a well-defined subset of a language's grammar and lexicon but adds the terminology needed in a technical domain [15]. Therefore, the concerned users are the experts who know the domain well and use simple and precise queries to query a given dataset. GFMed [7] is a controlled language system.

The major previous system approaches perform very accurate simple and compound queries. However, these systems fail when users want to filter the results by specific expressions containing differences, aggregations, ordering, negation, numbers, date comparisons, superlatives, and comparatives adjectives.

Inspired by this challenge, in this work, we propose a new system approach for querying Linked Data using different forms of NL queries. In particular, this system can deal with complex queries containing specific relations and can identify them and the links between their corresponding entities in the datasets.

3. Proposed System

To translate natural language queries into SPARQL, we believe that a querying process that consists of four steps should be followed, as shown in Figure 1.

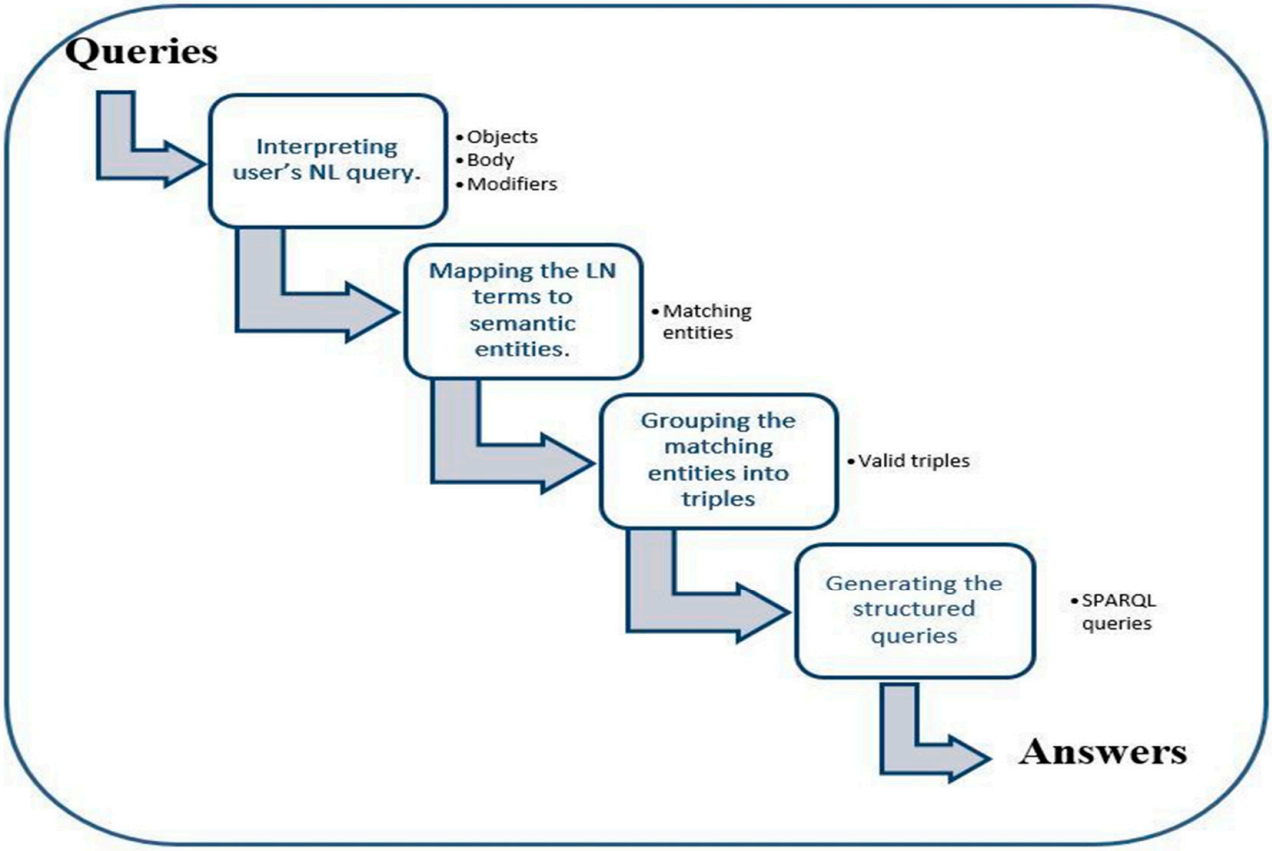

Figure 1. The proposed querying process.

In this context, users can explore their information needs more naturally without relying on technical aspects of the querying process. A detailed description of each step of our process with some illustrative examples is given in the following subsections.

3.1. Interpreting the User's NL Query

In order to understand the user need, we must identify the two main parts of a query, which are:

- The query object represents what the user wants in return.
- The query body consists of the semantic relations between the NL query terms.

We consider that combining these parts of the query supports the construction of a correct SPARQL query. For example, the query «give me all libraries established before 1400» has two main parts:

1. Libraries as a query object.

2. {libraries—established before—1400} as the query body.

While the user does not tell any information about these main parts in the asked query, we must process the syntactic structure of the query in order to infer these parts. However, the process of constructing the query to identify these parts is not a trivial task, especially for complex queries. In this context, we believe that the query interpretation can be performed using the representation of grammatical relations between words in a sentence provided by The Stanford CoreNLP [16] which is an NLP tool built by Stanford University and is one of the most widely used natural language analysis tools. CoreNLP enables users to derive linguistic annotations for text, including token and sentence boundaries, parts of speech, named entities, numeric and time values, dependency, and relations, as shown in Figure 2.

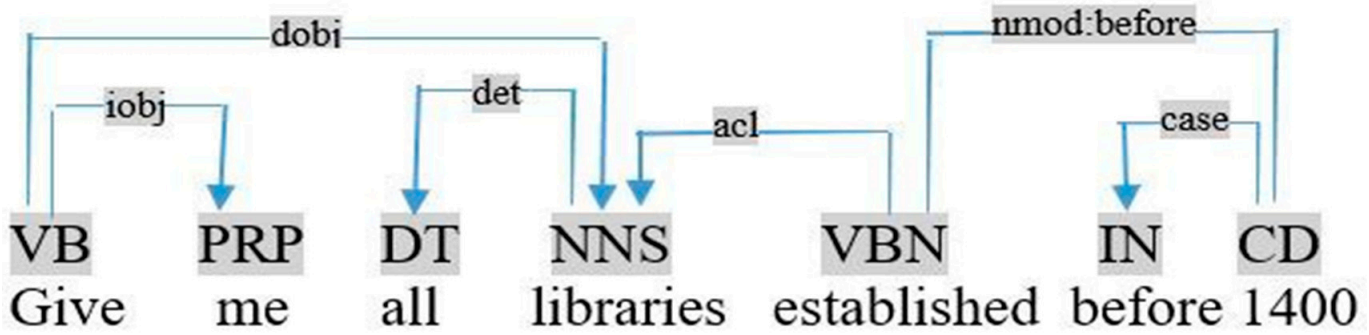

Figure 2. Graph dependency of the query «give me all libraries established before 1400».

3.1.1. Identifying the Query Class

In this section, we present the query structure as the sequence of the query's basic grammatical elements such as subject, verb, and so on. To identify the query structure in this context, we must process its word order, representing the specific sequence of words in the query. For example, the query «what is the profession of Frank Herbert» has the word order «WP - VBZ - DT - NN - IN - NNP - NNP» from the structure «Wh-word + Auxiliary/Modal verb + subject + rest of the sentence».

In our work, to identify the main query parts, we classify the queries into four types: yes/no questions, Wh questions, imperative queries, and compound queries. We note that this query classification is based on the grammar of the English language. In the following, we will describe the structure of each of these queries:

- Yes/no questions

The yes/no questions are made using auxiliary verbs (be, do, and have) or modal verbs (can, may, and will) at the beginning of a sentence. Especially, yes/no questions are the questions that can be answered by yes or no. Generally, these questions have the following structure:

Auxiliary/Modal verb + subject + verb + rest of the sentence (S1)

- Wh questions

Unlike yes/no questions, Wh-questions are used to ask for specific information. In particular, Wh-questions begin with what, when, where, who, whom, which, whose, why, and how. Generally, these questions have the following structures:

Wh-word + Auxiliary/Modal verb + subject + rest of the sentence (S2)

Wh-word + subject + Auxiliary/Modal verb + rest of the sentence (S3)

Wh-word + verb + rest of the sentence (S4)

- Imperative queries

Imperative queries are based on the structure of imperative sentences. They express an order or a request. Generally, they begin with verbs such as find, search, give, and show me.

The general structure of imperative queries is as follows:

$$\text{Verb + object + rest of the sentence} \quad \text{(S5)}$$

- Compound queries

Compound queries involve more than two simple queries connected by a coordinating conjunction (and, but, or).

$$\text{Query1 + coordinating conjunction + Query2} \quad \text{(S6)}$$

In practice, we apply Algorithm 1 to identify the query type as follows:

Algorithm 1: Identifying the query type

```
Input: q= {T1, T2,….TN}//list of query terms;
Output: q-type//the query type;
If (T1= Auxiliary/Modal verb) then
    q-type= "Yes/No question";
Else if (T1= Wh word) then //what, when, where, etc.
    q-type= "Wh question";
Else if ((T1= verb) then
    q-type= "imperative query";
Else
    q-type= "unknown";
End if
//check if q is compound
If  ∃ Ti∈{T1, T2,….TN} and Pos(Ti) ="cc" then//Ti is a coordinating conjunction
    q-type=  "compound   "+ q-type;
End if
Return q-type
```

3.1.2. Identifying the Query Object

As mentioned above, the query object is related to the query structure and the relations between words in the query. For this, we process each structure to identify its relative object. We apply Algorithm 2 to identify the question object of each structure.

Algorithm 2: Identifying the question object

```
Input: q= {T1, T2,….TN}//list of query terms;
SCDepLIST: list of Stanford CoreNLP dependencies of q
//each dependency has the form SCDep(Ti, Tj)
Output: Wh-objet//the question object;
Case 1: Yes/No questions (structure S1)
   Wh-objet = "Validation of question";
Case 2: Wh questions (structure S2 or S3)
  Case 2.1: Wh-word is Where
   Wh-objet = "Location";
  Case 2.2: Wh-word is When
   Wh-objet = "Date";
  Case 2.3: Wh-word is How many
    If( ∃ amod (Ti, many) ∈ SCDepLIST ) the
      If (∃ conj:and (Ti, Tk) ∈ SCDepLIST) then
          Wh-objet = count (Ti) + count (Tk);   //compound object
      Else
          Wh-objet = count (Ti);   //simple object
      End if
    Else If (∃ nsubj (Ti, Tj) ∈ SCDepLIST ) then
      If (∃ conj:and (Tj, Tk) ∈ SCDepLIST) then
          Wh-objet = count (Tj) + count (Tk);   //compound object
      Else
          Wh-objet = count (Tj) ;   //simple object
      End if
    End if
  Case 2.4: Wh-word different from how many
    // Tj is a nominal subject or passive nominal subject
    If ((∃ nsubj (Ti, Tj) ∈ SCDepLIST) or (∃ nsubjpass (Ti, Tj) ∈ SCDepLIST )) then
      If (∃ conj:and (Tj, Tk) ∈ SCDepLIST) then
          Wh-objet = Tj and Tk;   //compound object
      Else if (∃ conj:or (Tj, Tk) ∈ SCDepLIST) then
          Wh-objet = Tj or Tk;   //compound object
      Else
          Wh-objet = Tj ;//Tj   //simple object
      End if
    End if
Case 3: Wh questions (structure S4)
 Wh-objet= "variable"
Case 4: Imperative queries (structure S5)
    If (∃ dobj (Ti, Tj) ∈ SCDepLIST ) then
      If (∃ conj:and (Tj, Tk) ∈ SCDepLIST) then
          Wh-objet = Tj and Tk;   //compound object
      Else if (∃ conj:or (Tj, Tk) ∈ SCDepLIST) then
          Wh-objet = Tj or Tk;   //compound object
      Else
          Wh-objet = Tj ;   //simple object
      End if
    End if
 Return Wh-objet
```

In detail, when the user asks a yes/no question such as «is the wife of President Obama called Michelle», the object validates the query in the dataset. For example, the object is a verification if President Obama's wife is named Michelle or not.

Otherwise, when the user asks a Wh question, there are various cases (Structures (S2) and (S3)):

1. If the question word is "where" such as the query «where is Fort Knox located», the object here is a variable with category location.

2. If the question word is "when" such as the query «when did Michael Jackson die», the object here is a variable with category date.
3. If the question word is "how many", the query object is the quantity of the followed noun after "how many". There are two cases:
   (i) Many is an adjectival modifier of the followed noun, as shown in the dependency graph of the query «how many moons does Mars have», visualized in Figure 3.
   (ii) The following noun is a nominal subject of the auxiliary verb, as shown in the dependency graph of the query «how many awards has Bertrand Russell», visualized in Figure 4.

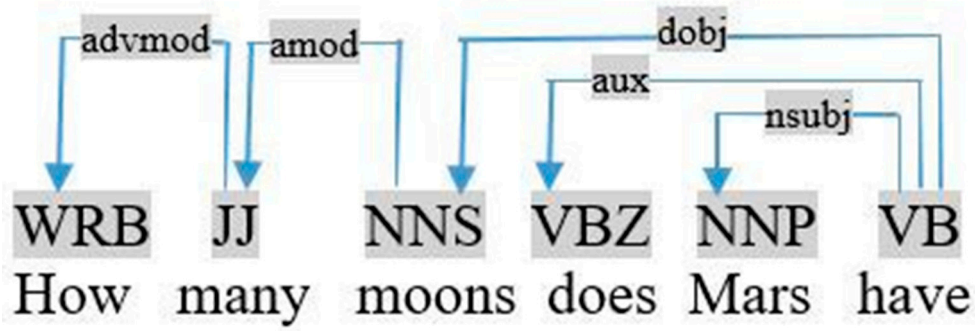

Figure 3. Graph dependency of the query «how many moons does Mars have».

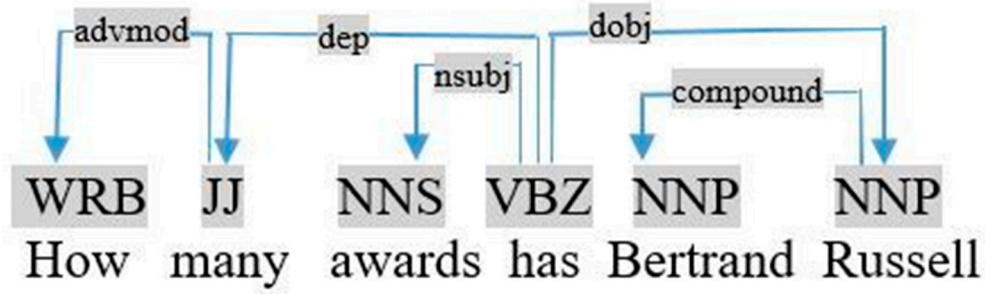

Figure 4. Graph dependency of the query «how many awards has Bertrand Russell».

– In the two previous cases, if the followed noun is connected to another noun by conjunction (Structure (S6)), as shown in Figure 5, the object is the sum of the quantities of the two nouns (e.g., count (rivers) + count (lakes)).

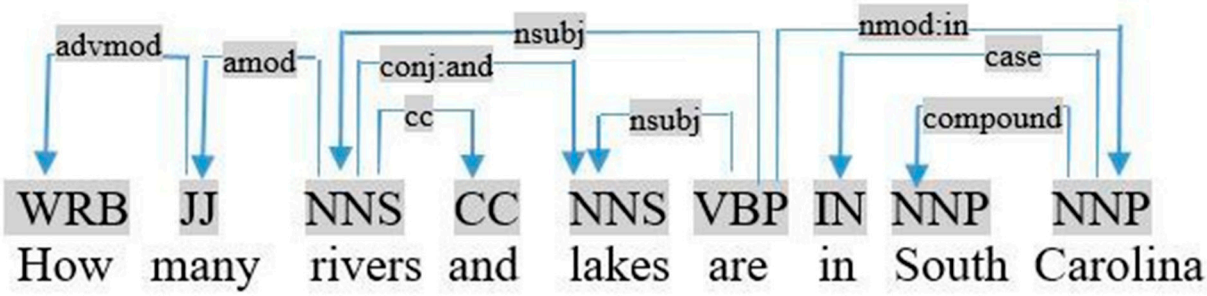

Figure 5. Graph dependency of the query «how many rivers and lakes are in South Carolina».

4. If the question word is different from "how many", there are two cases:
   (i) If the query is in the active mode, the object is the noun relative by the dependency nsubj, as shown in the dependency graph of the query «what is the profession of Frank Herbert», visualized in Figure 6.

(ii) If the query is in the passive mode, the object is the noun relative by the dependency nsubjpass, as shown in the dependency graph of the query «which software has been published by Mean Hamster Software», visualized in Figure 7.

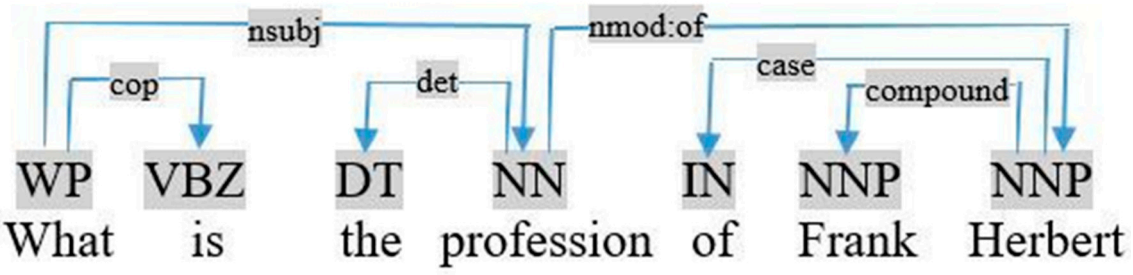

Figure 6. Graph dependency of the query «what is the profession of Frank Herbert».

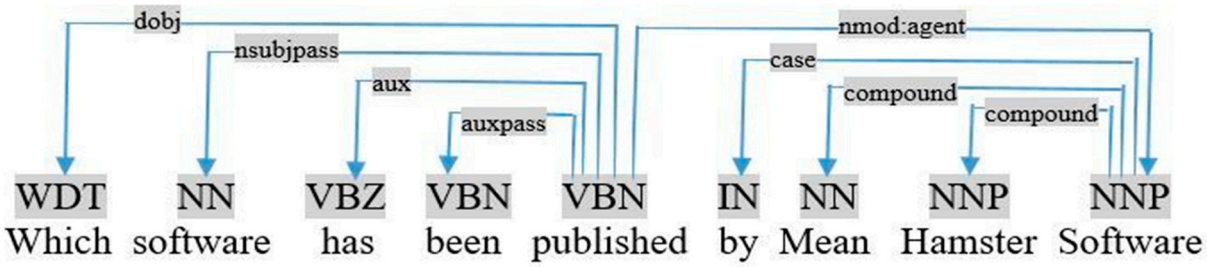

Figure 7. Graph dependency of the query «which software has been published by Mean Hamster Software».

– In the two previous cases, if the noun is connected to another noun by conjunction, as shown in Figure 8, the object is the coordination of two nouns (e.g., rivers and lakes).

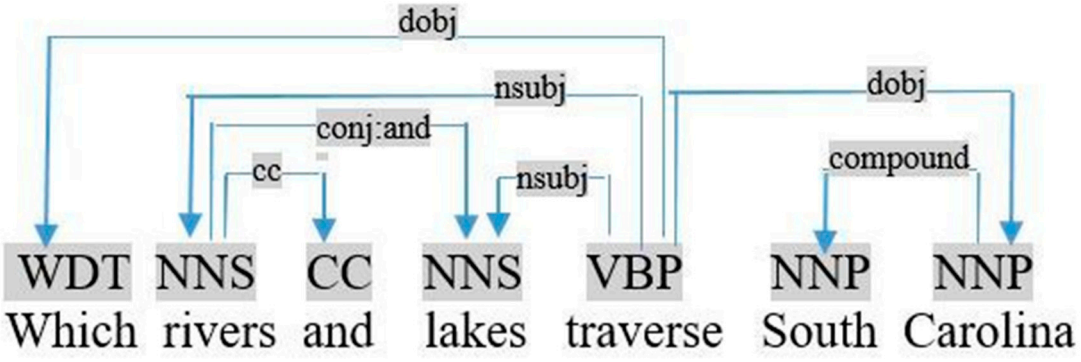

Figure 8. Graph dependency of the query «which rivers and lakes traverse South Carolina».

Otherwise, if the question word is subject, followed by the verb (Structure (S4)) and relative by the dependency nsubj, the object is a variable as shown in the dependency graph of the query «who created English Wikipedia», visualized in Figure 9.

Finally, when the user asks an imperative query (Structure (S5)), the object here is the noun relative by the dependency direct object dobj, as shown in the dependency graph of the query «give me all Argentine films», visualized in Figure 10.

– If this noun is connected to another noun by a conjunction, the object is the coordination of two nouns.

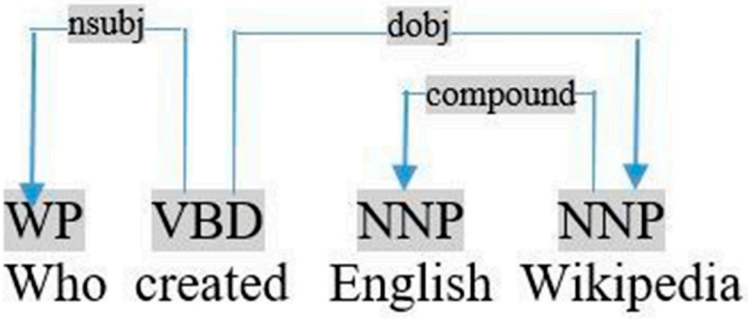

Figure 9. Graph dependency of the query «who created English Wikipedia».

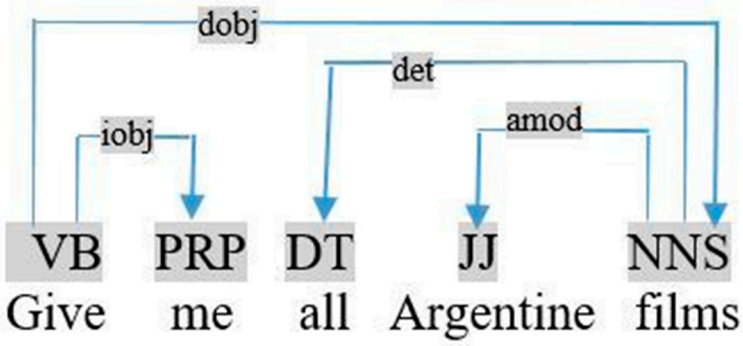

Figure 10. Graph dependency of the query «give me all Argentine films».

3.1.3. Identifying the Query Body

While the query body consists of the semantic relations between the NL query terms, we propose to extract relations between the query terms based on Stanford dependencies. In particular, we describe three forms of the relations between the query terms:

(a) A triple consisting of three parts, i.e., subject, predicate, and object.
(b) A relation consisting of two parts, i.e., Argument 1 and Argument 2.
(c) A restriction is composed of some conditions in the query.

For this, we use some proposed rules:

(1) Nominal subject:

$$\text{If } (\exists \text{ nsubj } (T_i, T_j) \in \text{SCDepLIST }) \Rightarrow \text{add (Relation } <T_j, T_i, \text{?var}>) \quad (1)$$

This rule is applied to extract a relation between each term T_i that has a nominal subject T_j in the query. For example, in Figure 3, (nsubj (have, Mars) ∈ SCDepLIST) ⇒ add (Relation <Mars, have, ?var>).

(2) Passive nominal subject

$$\text{If } (\exists \text{ nsubjpass } (T_i, T_{j)} \in \text{SCDepLIST}) \Rightarrow \text{add (Relation } <T_j, T_i, \text{?var}>) \quad (2)$$

This rule extracts a relation between each term T_i that has a passive nominal subject T_j in the query. For example, in Figure 7, (nsubjpass (published, software$_{)}$ ∈ SCDepLIST) ⇒ add (Relation <software, published, ?var>).

(3) Direct object

$$\text{If } (\exists \text{ dobj } (T_i, T_j) \in \text{SCDepLIST} \cap T_j \neq \text{Wh word}) \Rightarrow \text{add (Relation } <\text{?var}, T_i, T_j>) \quad (3)$$

This rule is applied to extract a relation between each term T_i that has a direct object T_j in the query. For example, in Figure 3, (dobj (have, moons) ∈ SCDepLIST) ⇒ add (Relation <?var, have, moons>).

(4) Adjectival modifier

$$\text{If } (\exists \text{ amod } (T_i, T_j) \in \text{SCDepLIST and } T_j! = \text{“many”}) \Rightarrow \text{add (Relation} < T_i, \text{unknown}, T_j >) \quad (4)$$

This rule is applied to extract a relation between each term T_i that has an adjectival modifier T_j that is different to many. For example, in Figure 10, **(**amod (films, Argentine) ∈ SCDepLIST) ⇒ add (Relation <films, unknown, Argentine>).

(5) Preposition

$$\text{If } (\exists \text{ nmod:X } (T_i, T_j) \in \text{SCDepLIST}) \Rightarrow \text{add (Triple} < T_i, X, T_k >) \quad (5)$$

This rule is applied when the relation between T_i and T_j is a preposition such as in, on, of, agent, as, by, to, and as. For example, in Figure 6, **(**nmod: of (profession, Herbert) ∈ SCDepLIST) ⇒ add (Triple <profession, of, Herbert>).

(6) Compound term

$$\text{If } \textbf{(}\exists \text{ compound } \textbf{(}T_i, T_j) \in \text{SCDepLIST}) \Rightarrow \forall\ T_i\ \textbf{replace (}T_i, T_j - T_i\textbf{)} \quad (6)$$

This rule replaces each term Ti with a compound term T_j in the query with the compound noun $T_i - T_j$. For example, **compound (**Russell, Bertrand) ⇒ **replace (**Russell, Bertrand-Russell).

(7) Negation

$$\text{If } (\exists \text{ neg } (T_i, \text{not}) \in \text{SCDepLIST}) \Rightarrow \forall\ T_i \text{ replace } (T_i, \text{not} - T_i) \quad (7)$$

This rule allows for the replacement of each term T_i preceded by not, by not $- T_i$. Consider the dependency graph of the query «what car is not made in Germany» as shown in Figure 11. This rule is used as follows:

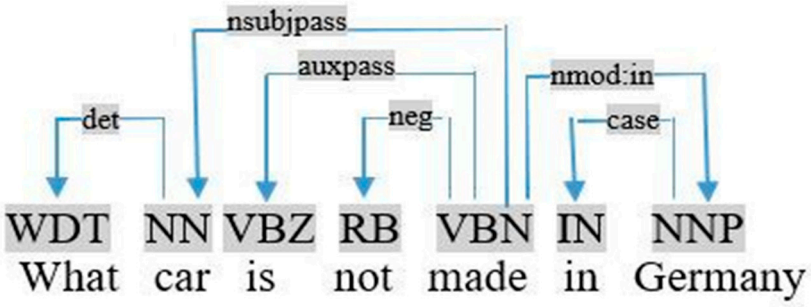

Figure 11. Graph dependency of the query «what car is not made in Germany».

neg (made, not) ⇒ **replace** (made, not-made)

(8) Optimization rules

In order to optimize some of the previous relations and identify the query triples, we propose the following specification rules:

$$\textbf{If (Relation} < T_j, T_i, \textbf{?var}> \cap \textbf{Relation} < ?\text{var}, T_i, T_k >\textbf{)} \Rightarrow \textbf{add (triple} < T_j, T_i, T_k > \cap \textbf{Delete (Relation} < T_j, T_i, \textbf{?var}>\textbf{)} \cap \textbf{Delete (Relation} < ?\text{var}, T_i, T_k >\textbf{)} \quad (8)$$

This rule is applied for each term T_i that has a *nominal subject* T_j and a *direct object* T_k in the query. For example,

Relation <Mars, have, **?var>** ∩ **Relation <?var**, have, moons**>** ⇒ **add (triple <**Mars, have, moons**>** ∩ **Delete (Relation <**Mars, have, **?var>)** ∩ **Delete (Relation <?var**, have, moons**>)**

$$\textbf{If (Relation} < T_j, T_i, \textbf{?var}> \cap \textbf{Triple} < T_i, X, T_k >\textbf{)} \Rightarrow \textbf{add (triple} < T_j, T_i\text{-}X, T_k > \cap \textbf{Delete (Relation} < T_j, T_i, \textbf{?var}>\textbf{)} \cap \textbf{Delete (Triple} < T_i, X, T_k >\textbf{)} \quad (9)$$

This rule is applied for each term T_i with a nominal or passive nominal subject T_j and relative by a preposition with a term T_k in the query. For example,

Relation <rivers, are, ?var> ∩ Triple <are, in, Carolina> add (triple <rivers, are-in, Carolina> ∩ Delete (Relation Relation <rivers, are, ?var>) ∩ Delete (Triple <are, in, Carolina**>)**

(9) Comparative adjectives

Generally, comparative adjectives are used to compare one person or thing with another. In the following, we describe the proposed rules of some comparative adjectives used in the NL queries:

- **Same** This adjective indicates that two or more things are exactly like one another. Especially, we use "same" as an adjective before a noun in two cases:
  - (i) With "as" to compare two nouns in a simple query. In practice, we applied the following rule:

$$\begin{gathered}\text{If (triple} <T_i, T_j, T_k> \cap \text{Relation} <T_k, \text{unknown, same}> \cap \text{triple} <T_k, \text{as}, T_s>) \Rightarrow \\ \text{add triple} <T_s, T_j, T_k> \cap \text{Delete (Relation} <T_k, \text{unknown, same}>) \cap \text{Delete (triple} <T_k, \text{as}, T_s>)\end{gathered} \quad (10)$$

This rule is applied when the predicate T_j relates the two terms T_i and T_s to the same term T_k in the query. Therefore, we add a new triple to describe the relation between T_s and T_k. For example, in Figure 12, "artists" and "Rachel Stevens" are related by "born-on" to the same term "date" in the query «which artists were born on the same date as Rachel Stevens». Therefore, we add a new triple as follows: triple <artist, born-on, date> ∩ Relation <date, unknown, same> ∩ triple <date, as, Rachel- Stevens> ⇒ add triple <Rachel- Stevens, born-on, date> ∩ Delete (Relation <date, unknown, same>) ∩ Delete (triple <date, as, Rachel-Stevens>).

  - (ii) To compare two nouns in a complex query. In practice, we apply the following rule:

$$\begin{gathered}\text{If (triple} <T_i, T_j, T_k> \cap \text{Relation} <T_k, \text{unknown, same}> \cap \text{triple} <T_s, T_r, T_k>) \Rightarrow \\ \text{add triple} <T_i, T_j, \text{var1}> \cap \text{add triple} <T_s, T_r, \text{var2}> \cap \text{Delete (Relation} <T_k, \text{unknown, same}>) \cap \\ \text{Delete (triple} <T_i, T_j, T_k>) \cap \text{Delete (triple triple} <T_s, T_r, T_k>) \cap \text{add restriction (var1 = var2)}\end{gathered} \quad (11)$$

This rule is applied when two terms, T_i and T_s, are related to the same term T_k, by different relations: T_j and T_r. We replace T_k in the two triples with variables: **?**var1 and ?var2. We also add a restriction, indicating that "var1" and "var2" are equals. For example, in Figure 13, daughters and they (daughters) are related to the same term "place" by "died-at" and "born-at" in the complex query «which daughters of British earls died at the same place they were born at». Therefore, we add two triples and a restriction as follows: triple <daughters, died-at, place> ∩ Relation <place, unknown, same> ∩ triple <they, born-at, place> ⇒ add triple <daughters, died-at, ?var1> ∩ add triple <they, born-at, ?var2> ∩ Delete (triple <daughters, died-at, place>) ∩ Delete (Relation <place, unknown, same>) ∩ Delete (triple <they, born-at, place>) ∩ add restriction (?var1 = ?var2).

- **More/less**

These **adjectives** indicate whether a person or thing has more or less of a certain quality. In particular, more and less are generally used in the following query structure:

Wh word +noun (subject) + verb + more/less + than + number + noun (object)

More indicates that the quantity of the noun after more than is larger than the given number. Otherwise, less indicates that the noun's quantity after less than is smaller than the given number. In practice, we applied the two following rules:

If (advmod (T_i, more) ∩ mwe (more, than) ∩ nummod (T_j, T_i)) ⇒ restriction (count (T_j) > T_i) (12)

If (advmod (T_i, less) ∩ mwe (less, than) ∩ nummod (T_j, T_i))⇒ restriction (count (T_j) < T_i) (13)

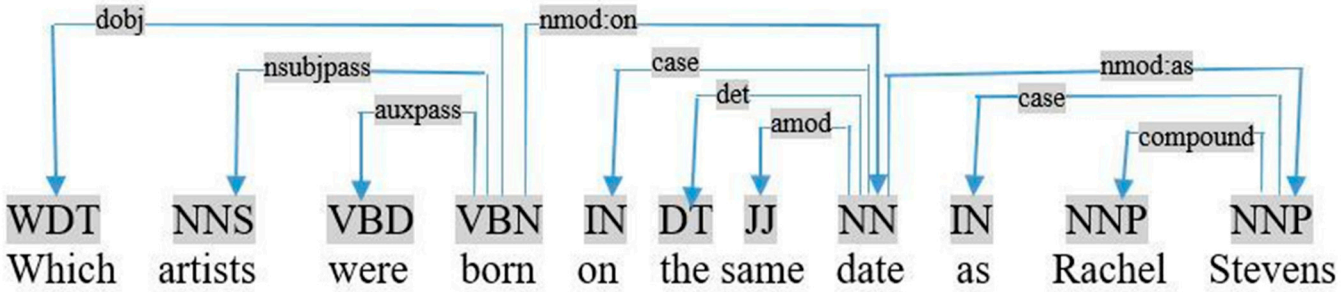

Figure 12. Graph dependency of the query «which artists were born on the same date as Rachel Stevens».

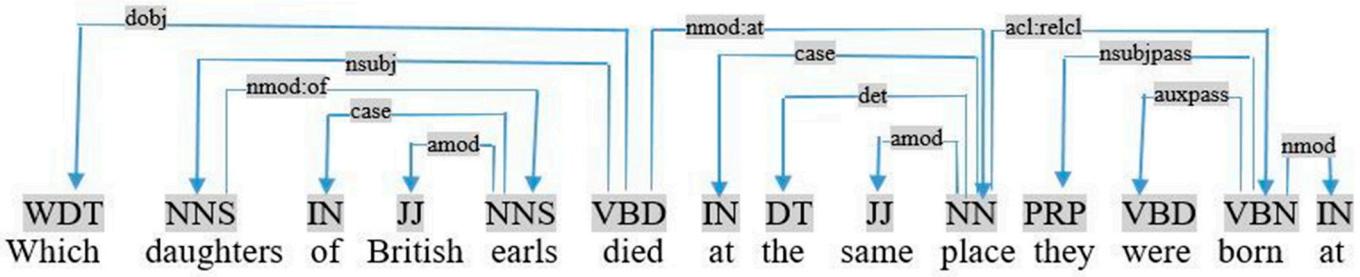

Figure 13. Graph dependency of the query «which daughters of British earls died at the same place they were born at».

For example, in Figure 14, the query «which countries have places with more than two caves» gives the following restriction:

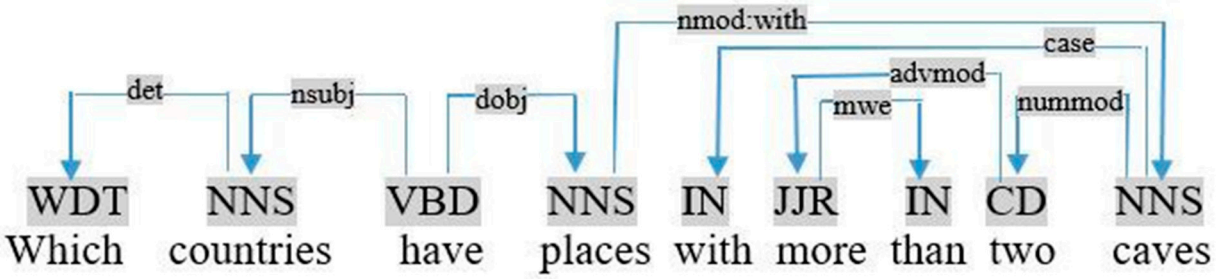

Figure 14. Graph dependency of the query «which countries have places with more than two caves».

Advmod (two, more) ∩ mwe (more, than) ∩ nummod (caves, two) ⇒ restriction (count(caves), >, two)

(10) Superlative adjectives Superlative adjectives describe a person or thing at the upper or lower limit of a quality relative to all other people or things in a group. Superlative adjectives are generally used in the following query structure:

WH word + noun (subject) + verb + the + superlative adjective + noun (object)

These adjectives have three forms:

(i) Adjective + est, such as "highest" in the query «which U.S. state has the highest population density», as shown in Figure 15. In this case, we applied the following rule:

If (Relation <T_i, unknown, T_j> ∩ Pos (Tj) = JJS) ⇒ restriction (T_i, Sup, T_j) (14)

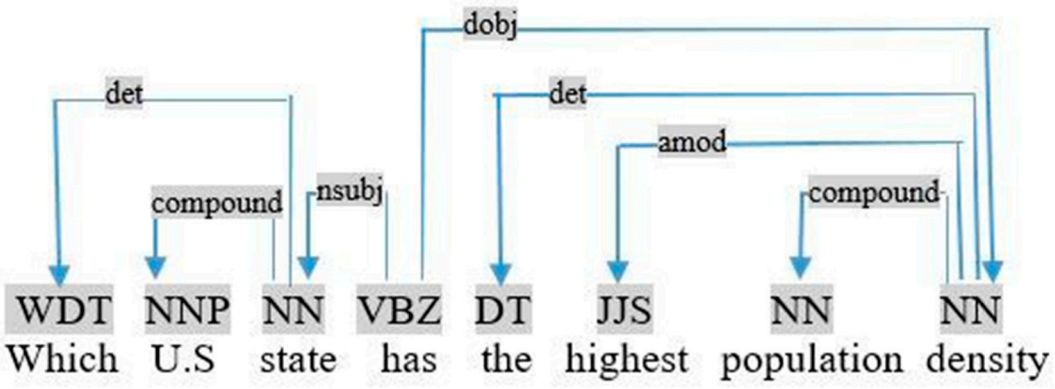

Figure 15. Graph dependency of the query «which U.S. state has the highest population density».

This rule identifies the relationship between the superlative adjective Tj and the noun Ti. Consider the previous query, this rule is used to generate the following restriction: Relation <density, unknown, highest> ∩ Pos (highest) = JJS ⇒ restriction (density, Sup, highest)

(ii) most/least + adjective, such as "most deep" in the query «where is the most deep point in the ocean», as shown in Figure 16. In this case, we applied the following rule:

If (advmod (T_i, most/least) ∩ Relation <T_j, unknown, T_i> ∩ Pos (Ti) = JJ) ⇒ restriction (T_j, Sup, most/least-T_i) (15)

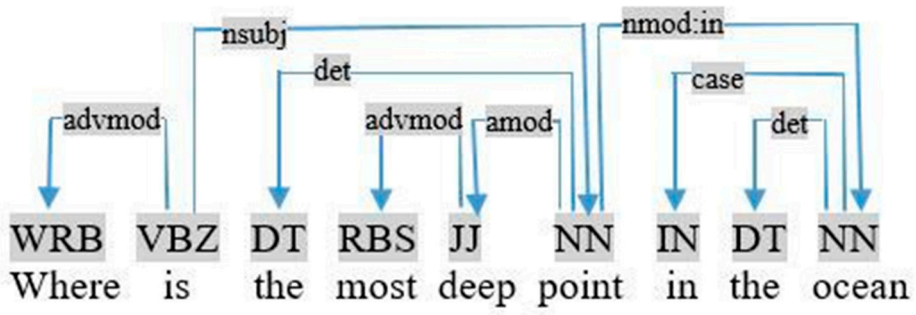

Figure 16. Graph dependency of the query «where is the most deep point in the ocean».

This rule identifies the relationship between the most/least Ti superlative adjective and the noun Tj. Consider the previous query, this rule is used to generate the following restriction:

advmod (deep, most) ∩ Relation <point, unknown, deep> ∩ Pos (deep) = JJ ⇒ restriction (point, Sup, most-deep)

(iii) Irregular adjective, such as "most" in the query «which book has the most pages», as shown in Figure 17. In this case, we applied the following rule:

If (Relation <T_i, unknown, most/least> ⇒ restriction (T_i, Sup, most/least) (16)

This rule identifies the relationship between the superlative adjective most/least and the noun Ti. Consider the previous query, this rule is used to generate the following restriction:

If (Relation <pages, unknown, most> ⇒ restriction (pages, Sup, most)

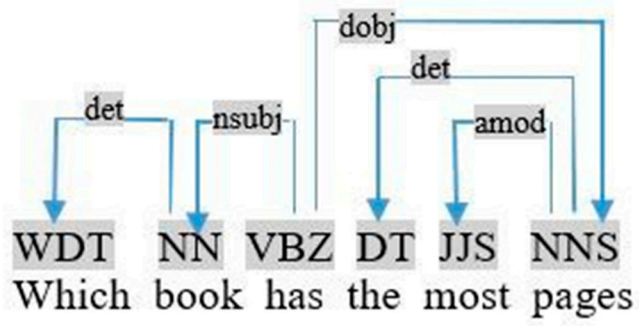

Figure 17. Graph dependency of the query «which book has the most pages».

(11) Numbers

In addition to the numbers related to the comparative adjectives, a number may be attached to a given noun in the query as a quantifier. In this case, we applied the following rule:

$$\text{If (nummod (Ti, Tj))} \Rightarrow \text{restriction (limit, Tj, Ti)} \quad (17)$$

This rule indicates that the cardinality value Tj is related to the noun Ti. For example, in the query with a quantifier «what 2 cars are from Germany», the cardinality value 2 is related to the noun cars, as visualized in Figure 18. Therefore, this rule is used to generate the following restriction:

$$\text{nummod (cars, 2)} \Rightarrow \text{restriction (limit, 2, cars)}$$

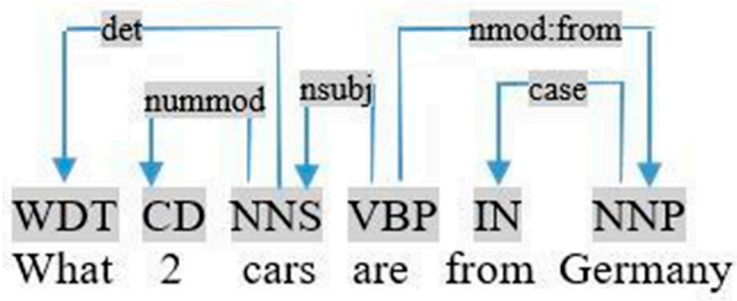

Figure 18. Graph dependency of the query «what 2 cars are from Germany».

3.2. Mapping the NL Terms to Semantic Entities

In this second step, we try to match each term in the query to one or more entities in the chosen dataset without considering their relations. In particular, to realize this mapping of NL terms to the entities in the dataset, we use the DBpedia Lookup (https://www.dbpedia.org/resources/lookup/ (accessed on 20 December 2022), an entity retrieval service for Linked Data. This service can extract all the related entity identifiers in DBpedia [17] for a given term, one of the most famous steps of the decentralized Linked Data effort.

Next, to classify the matching entities, we rank them by the Levenshtein (http://www.Merriampark.com/ld.htm, accessed on 18 April 2007.) distances between the query term and these entities. Then, we eliminate all the matching entities with a Levenshtein distance greater than 5 to select the closest entities of the initial term. When this term has no appropriate matching entities in DBpedia, we have to search for its synonyms in WordNet [14]. If this term or at least one of its synonyms has matching entities in DBpedia, the list of matching entities consists of the mapped entities ranked with the Levenshtein distances. Otherwise, if a query term or its synonyms in WordNet do not have matching entities in DBpedia, we establish a dialogue with the user to get clarification. For example, in the query «how many rivers and lakes are in South Carolina», we obtained the matching entities listed in Table 1.

Table 1. Mapping of NL terms to DBpedia entities.

Query Terms	Matching Entities in DBpedia	Uri	Levenshtein Distance
Rivers	River	http://dbpedia.org/ontology/River	2
Lakes	Lake	http://dbpedia.org/resource/Lake	2
	Place	http://dbpedia.org/ontology/Place	3
South-Carolina	South_Carolina	http://dbpedia.org/resource/South_Carolina	1

After identifying the matching entities of each term, we combine them to identify all possible queries. Finally, these queries are classified in ascending order of the averages of the Levenshtein distances associated with each entity belonging to a given query (Table 2).

Table 2. Query classification.

Queries	Average of Levenshtein Distances
<dbo (http://dbpedia.org/ontology/): River, unknown, dbr (http://dbpedia.org/resource/): South_Carolina> ∩ <dbr: Lake, unknown, dbr: South_Carolina>	1.66
<dbo: River, unknown, dbr: South_Carolina> ∩ < dbo: Place, uknown, dbr: South_Carolina>	2

3.3. Grouping the Mapped Semantic Entities into Valid Triples

The third step needs to represent the query in the form of triples because triples are the fundamental data structure of RDF. This requires using the identified semantic relations, the matching entities of NL terms, and the dataset structure to construct valid triples. In this context, we try to exploit the existing semantic relations between the different entities of the queried dataset to build valid triples that represent the semantics of the query (Table 3).

Table 3. Triple validation.

Initial Relation	SPARQL Query	Result	Valid Triple
Relation <Argument1, unknown, Argument2>	**Query 1:** Select distinct ?prop where { ?uri rdf:type prefix: Argument1. ?uri ?prop prefix: Argument2. }	Prop	<Argument1, **Prop**, Argument2>
Relation <Argument1, relation, ?var>	**Query 2:** Select distinct ?Arg2 where {?uri rdf:type prefix: Argument1. ?uri prefix:relation ?Arg2. }	Arg2	<Argument1, relation, **Arg2**>
Relation < ?var, relation, Argument2>	**Query 3:** Select distinct ?Arg1 where { ?uri rdf:type prefix: Argument2. ?Arg1 prefix: relation ?Argument2. }	Arg1	<**Arg1**, relation, Argument2>
triple<Argument, relation, Argument>	**Query1** Else **Query2** Else **Query3**	Prop Else Arg2 Else Arg1	<Argument1, **Prop**, Argument2> **Else** <Argument1, relation, **Arg2**> **Else** < **Arg1**, relation, Argument2>

First, we look for the relations between each pair of arguments in the triples with an unknown relation <Argument1, unknown, Argument2>. In particular, we execute the

SPARQL query Query1 to find the unknown property between two arguments (Argument 1 and Argument 2). For example, in the previous query, we execute the SPARQL query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?prop where {
?uri rdf:type dbo:River.
?uri ?prop res:South_Carolina.
```

− If this SPARQL query has a result, we confirm that the two arguments are related, and we complete the triple with the relationship found. For example, dbo:location is the result of the previous SPARQL query. Therefore, the valid triple is: < dbo: River, dbo:location, dbr: South_Carolina>.

Next, we examine the triples of the form <Argument1, relation, ?var> and <?var, relation, Argument2 > to check whether the argument is the domain or the range of the relation and to enrich the query with the unknown argument, which can be a class, instance, or literal. In particular, we run the SPARQL queries Query2 and Query3 to find the missing argument.

Finally, we treat the triples in the form <Argument, relation, Argument> by executing one of the three previous SPARQL queries to identify a valid triple.

This step results in the queries containing all valid triples to generate correct SPARQL queries.

3.4. Generating the SPARQL Query and Extracting the Answers

SPARQL (SPARQL Protocol and RDF Query Language) [3] is the query language used to query semantic web data in RDF. Generally, the SPARQL query has the following syntax:

```
Prefix declarations>
SELECT <query-objects>
WHERE <query-body>
<Query modifiers>
```

(1) Prefix declarations

The prefix declarations represent the abbreviations of the URIs. In the following, we describe the principal prefixes used in this work:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbr: http://dbpedia.org/resource/>
PREFIX dbc: http://dbpedia.org/resource/Category:
PREFIX yago: <http://dbpedia.org/class/yago/>
```

(2) Query objects

In SPARQL language, the query objects represent the variables prefixed with "?" that will appear in the query results. In particular, we have some cases:

- If the query object is an entity of type RDF class, we interpret it as a variable prefixed with "?"after SELECT. We also interpret it in the query body as a triple pattern <?class-Name rdf:type ClassName> after the "WHERE" clause. For example, the following query returns all instances of the class dbo:River:

```
SELECT DISTINCT ?river WHERE {
?river rdf:type dbo:River.}
```

- If the query object is an entity of type RDF literal, we interpret it directly as a variable prefixed with "?" after SELECT. Similarly, we interpret it in the query body as a

triple pattern <?x, DataProperty, ?L> after the "WHERE" clause, as shown in the following example:

```
SELECT DISTINCT ?title WHERE {
?book rdf:type dbo:Book.
?book rdfs:label ?title}
```

- If the query object is a variable, we interpret it as a variable "?uri" in the head and the body of the query, as shown in the following example:

```
SELECT ?uri
WHERE {
dbr:Wikipedia dbo:author ?uri}
```

- If the query object is a counted variable, we interpret it as the variable COUNT (?var) after SELECT, as shown in the following example:

```
SELECT (COUNT(?Awards) AS ?Counter)
WHERE {
dbr:Bertrand_Russell dbp:awards ?Awards}
```

- If the query object is the validation of the query in the dataset, we use ASK queries which intuitively correspond to a yes/no question in conversational language. For example, the previous query «is the wife of President Obama called Michelle» corresponds to a yes/no question, and has the following SPARQL query:

```
ASK WHERE {res:Barack_Obama dbo:spouse ?spouse. ?spouse rdfs:label ?name FILTER regex
(?name, \"Michelle\")
```

(3) Query body

The body of the query consists of all the valid triples. The instances and the properties are interpreted as their URIs. In contrast, we interpret the classes and the RDF literal entities as variables according to the previous rules.

(4) Query modifiers

The query modifiers represent the restrictions of the query that can be interpreted with the SPARQL language. In particular, SPARQL has some functions that restrict the solutions of a graph pattern match according to a given expression. In the following, we describe some SPARQL functions:

- FILTER (condition) is a clause inserted into the SPARQL query to filter the results. The condition inside the parentheses is a Boolean-type expression, and only those results where the expression returns true are used. FILTER functions can test the values of RDF literal strings. They can also restrict the numeric values and the dates by arithmetic expressions. For example, in the previous query «which daughters of British earls died at the same place they were born at», we interpret the restriction: Restriction (?var1 = ?var2) as follows:

```
SELECT DISTINCT ?uri
WHERE {?uri rdf:type yago:WikicatDaughtersOfBritishEarls;
?uri dbo:birthPlace ?var1.
?uri dbo:deathPlace ?var2.
FILTER (?var1 = ?var2)}
```

− We note that the negation is a special filter function by providing the operator "!" and the instruction "NOT EXISTS".

```
SELECT ?birth ?death ?person WHERE {?person dbo:birthPlace dbr:Berlin.
?person dbo:birthDate ?birth.
?person dbo:deathDate ?death.
FILTER NOT EXISTS {?person dbo:deathDate dbr:Berlin}}
```

- HAVING (condition) is analogous to a FILTER expression but operates over groups rather than individual solutions. HAVING is always used with GROUP BY, a clause that groups query solutions according to one or more expressions. For example, in the query «which countries have places with more than two caves», we interpret the restriction: Restriction (count (caves), >, two) as follows:

```
SELECT DISTINCT ?uri
WHERE {?uri rdf:type dbo:Country.
?cave rdf:type dbo:Cave.
?cave dbo:location ?uri. }
GROUP BY ?uri HAVING (COUNT(?cave) > 2)
```

- The LIMIT number clause puts an upper bound on the number of solutions returned. If the number of actual solutions is greater than the limit, then at most, the limit number of solutions will be returned.
- The ORDER BY ASC/DESC clause establishes the order of a solution sequence. Each ordering comparator is either ascending, indicated by the ASC() modifier, or descending, indicated by the DESC() modifier. Generally, this clause is used to interpret the superlative adjectives as shown in the query «which book has the most pages»:

```
SELECT DISTINCT ?uri
WHERE { ?uri rdf:type dbo:Book.
?uri dbo:numberOfPages ?n.}
ORDER BY DESC(?n) LIMIT 1
```

Thus, we need to combine the prefix declarations, query objects, valid triples, and query restriction to generate the final SPARQL query that will be executed to return the relevant responses.

4. Evaluation

The system was evaluated against test questions of the QALD-9 challenge [18]. We used precision, recall, and F-measure metrics to measure the performance of the proposed system. From 150 queries of the QALD-9 test, we identified 50 as complex queries representing the compound queries and queries containing numbers, date comparisons, superlatives, and comparatives adjectives. As described in Table 4, our system correctly answered 110 queries among 150, for which 80 simple and 30 complex queries were analyzed correctly.

Table 4. QALD-9 test dataset results.

Queries	Total	Processed	Correct	Precision	Recall	F-Mesure
Simple	100	95	80	0.84	0.8	0.82
Complex	50	40	30	0.75	0.6	0.66
Total	150	135	110	0.81	0.73	0.76

The experimental results show that the proposed system had the best results compared to the state-of-the-art precision, recall, and F-measure methods. Table 5 shows a comparison of the performance of the proposed system and some existing systems.

Table 5. Comparison of the proposed system with existing systems.

System	Challenge	Precision	Recall	F-Mesure
FREyA [5]	QALD-1	0.63	0.54	0.58
DEANNA [9]	QALD-3	0.21	0.21	0.21
GAnswer [9]	QALD-3	0.40	0.40	0.40
Casia [8]	QALD-4	0.32	0.40	0.36
WDAqua [11]	QALD-7	0.63	0.32	0.42
Proposed system	QALD-9	0.81	0.73	0.76

However, our system failed to parse some queries correctly. The following reasons can explain these failures:

(1) The system fails to find the relations between the terms of some complex queries, such as «show me hiking trails in the Grand Canyon where there is no danger of flash floods». Therefore, this leads to misinterpretation of queries.
(2) The system cannot find matching entities of a given term. For example, the term "grand-children" in the query «how many grand-children did Jacques Cousteau have» has no matching entities. Thus, the system must learn this kind of term in another way: uri1 child of uri2 and uri2 child of uri3.
(3) Finally, the system cannot express some NL queries using the SPARQL language, such as the query «who became president after JFK died». It implies that we are limited by formal language syntax, which can produce irrelevant answers.

5. Discussion

Most users are unfamiliar with formal query languages such as SPARQL, which precisely describes their information needs in the form of queries. As a result, they may prefer to express their intentions in simple languages such as NL queries. However, given the specified representation of data in the dataset, existing query systems face a significant challenge in understanding NL queries, particularly complex queries.

The primary goal of this paper is to present a new approach for querying Linked Data with NL queries, particularly complex queries containing negation, numbers, superlatives, and comparative adjectives. Our proposed system differs from others in that it focuses on the semantic relations between NL query terms, which can be identified using the representation of grammatical relations between words in a query. We use some proposed rules to describe the particular triples, relations, and query restrictions.

Following that, we identify each term's matching entities and use the existing link between the different entities of the queried dataset to construct the query's valid triples. Finally, we generate the final SPARQL query that will be executed to return the appropriate results. However, our system could not express some NL queries in SPARQL. This finding suggests that the ambiguity of the NL queries and the syntax of the SPARQL language impact the quality of answers and the performance of the proposed system.

6. Conclusions

Using natural language queries to query Linked Data provides novice users with a simple and natural way to access multiple heterogeneous semantic sources and datasets. However, the ambiguity causes significant issues when analyzing NL queries and translating them into SPARQL queries to return precise answers.

Numerous matching entities are returned, in particular, to interpret an ambiguous word in the query. In this context, the disambiguation of a given word in the query can influence the mapping of other words and the semantics of query. Furthermore, the syntactic relations between these words do not always correspond to semantic relations in queried datasets. As a result, mapping NL terms to semantic entities may result in multiple interpretations of the meaning of an NL query, with some interpretations relevant and others not relevant.

In this study, we propose to assist users in asking questions in NL using their terminology and receiving relevant answers by querying Linked Data knowledge. To address the issue of ambiguity in NL queries, we must consider the semantic relations between NL query terms during the query interpretation process. We intend to improve our system in the future to use other datasets and to handle more complex queries.

Author Contributions: Conceptualization, H.B. and Z.B.; methodology, H.B.; software, H.B.; validation, H.B.; writing—review and editing, H.B. and Z.B.; supervision, Z.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable, the study did not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bizer, C.; Heath, T.; Berners-Lee, T. Linked Data—The Story So Far. *J. Semant. Web Inf. Syst.* **2009**, *5*, 1–22. [CrossRef]
2. Linked Data. Available online: http://www.w3.org/DesignIssues/LinkedData.html (accessed on 18 June 2009).
3. SPARQL Query Language for RDF. Available online: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ (accessed on 26 March 2013).
4. Moussa, A.M.; Abdel-Kader, R.F. QASYO: A Question Answering System for YAGO Ontology. *Int. J. Database Theory Appl.* **2011**, *4*, 99–112.
5. Damljanovic, D.; Agatonovic, M.; Cunningham, H. FREyA: An Interactive Way of Querying Linked Data Using Natural Language. In Proceedings of the 1st Workshop on Question Answering over Linked Data (ESWC 2011), Heraklion, Greece, 30 May 2011. [CrossRef]
6. Yahya, M.; Berberich, K.; Elbassuoni, S.; Ramanath, M.; Tresp, V.; Weikum, G. Natural Language Questions for the Web of Data. In Proceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP CoNLL 2012), Jeju Island, Republic of Korea, 12–14 July 2012.
7. Marginean, A. GFMed: Question answering over biomedical linked data with grammatical framework. In Proceedings of the 1st International Workshop on Natural Language Interfaces for Web of Data (NLI-WoD 2014) co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, 19–23 October 2014. [CrossRef]
8. Shizhu, H.; Yuanzhe, Z.; Liu, K.; Zhao, J. CASIA@V2: A MLN-based question answering system over linked data. In Proceedings of the Working Notes for {CLEF} 2014 Conference (CLEF 2014), Sheffield, UK, 15–18 September 2014.
9. Zou, L.; Huang, R.; Wang, H.; Xu, Y.J.; He, W.; Zhao, D. Natural langauge question answering over RDF a graph data driven approach. In Proceedings of the SIGMOD '14:International Conference on Management of Data, New York, NY, USA, 22–27 June 2014. [CrossRef]
10. Xu, K.; Feng, Y.; Zhao, D. Xser@QALD-4: Answering natural language questions via phrasal semantic parsing. In Proceedings of the Working Notes for {CLEF} 2014 Conference (CLEF 2014), Sheffield, UK, 15–18 September 2014. [CrossRef]
11. Diefenbach, D.; Deep Singh, K.; Maret, P. Wdaqua-core0: A question answering component for the research community. In Proceedings of the SemWebEval 2017, Portoroz, Slovenia, 28 May–1 June 2017. [CrossRef]
12. Zheng, W.; Yu, J.X.; Zou, L.; Cheng, H. Question Answering Over Knowledge Graphs: Question Understanding Via Template Decomposition. *Proc. VLDB Endow.* **2018**, *11*, 1373–1386. [CrossRef]
13. Kotnis, B.; Lawrence, C.; Niepert, M. Answering Complex Queries in Knowledge Graphs with Bidirectional Sequence Encoders. In Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21), Vancouver, BC, Canada, 2–9 February 2021. [CrossRef]
14. Miller, G.A.; Beckwith, R.; Fellbaum, C.; Gross, D.; Miller, K. Introduction to WordNet: An on-line lexical database. *Int. J. Lexicogr.* **1990**, *3*, 235–244. [CrossRef]
15. Kittredge, R.I. Sublanguages. In *The Oxford Handbook of Computational Linguistics*; Mitkov, R., Ed.; Oxford University Press, Inc.: New York, NY, USA, 2003; pp. 430–447. [CrossRef]
16. Manning, C.D.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, R.S.; McClosky, D. The stanford corenlp natural language processing toolkit. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Stanford University, Baltimore, MD, USA, 22–27 June 2014. [CrossRef]
17. Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z. Dbpedia: A nucleus for a web of open data in The semantic web. In Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Republic of Korea, 11–15 November 2007. [CrossRef]
18. Usbeck, R.; Gusmita, R.H.; Ngomo, A.N.; Saleem, M. 9th Challenge on Question Answering over Linked Data (QALD-9). In Proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data Challenge (QALD-9) Co-Located with 17th International Semantic Web Conference (ISWC 2018), Monterey, CA, USA, 8–9 October 2018.