Template-Based Question Answering System Over the Semantic Web

Aarthi Dhandapani, VIT University, Chennai, India*
Viswanathan Vadivel, VIT University, Chennai, India

ABSTRACT

A question answering system is the most promising way of retrieving data from the available knowledge base to the end-users to get the appropriate result for their questions. Many question answering systems convert the questions into triples which are mapped to the knowledge base from which the answer is derived. However, these triples do not express the semantic representation of the question, due to which the answers cannot be located. To handle this, a template-based approach is proposed that classifies the question types and finds appropriate SPARQL query template for each type including comparatives and superlatives. The SPARQL query built is executed in the DBpedia endpoint and results are obtained. Compared with other factoid question answering systems, the proposed approach has the potential to deal with a large number of question types, including comparatives and superlatives. Also, the experimental evaluations of the system performed on the QALD 8 dataset presents good performance and can help users to find answers to their questions.

KEYWORDS

DBpedia, Knowledge Base, Natural Language Processing, Question Answering, SPARQL

1 INTRODUCTION

Question Answering (QA) is the fusion of natural language processing (NLP), Machine Learning (ML) and Semantic Analysis. It is used everywhere in various domains such as medical, education systems, personal assistants. In the last decade, there has been significant growth in a new part of the internet, namely the semantic web. The semantic web is developed with the motive to link the data available across multiple web pages, organize them in such a way that the data is directly readable by machines. It includes data sources in different forms. The structured database contains information that is organized and easy to access.

A substantial increase in the amount of research work over semantic web data creates an interaction model that enables people to benefit from semantic web standards. The QA process over dynamic data sources appears to be the most optimistic method to access information. At the same time, Question Answering systems hide their complexity behind an intuitive interface that is easy to use. The spontaneous increase in the semantic web data has resulted in heterogeneous data, as a result of this many systems has to compete with the types, quantity, of data sources.

Most Semantic Web Resources use RDF Query language to access the database, which implies that the user has to learn a query language to scan the semantic web. To facilitate that, the developed
Question Answering System that helps people to retrieve data from the database they want. It can be useful in preserving their private Knowledge Bases (KB) within the organizations. They can trust the system to access the data sources, other than depending on additional services. The Question Answering system can enact any data source, function better because it has developed to manage independent and domain-dependent applications. It varies depending on the types of inputs, maybe a keyword or a description. It also requires factoid, affirmation type of questions, and have a clear understanding, reasoning of reality, the origins of data from various domains. Thus, the scope of QA is enormous and widely acceptable.

In Real time application, the Text Retrieval Conference (TREC) (Voorhees, 2001) was the first large scale evaluation of domain-independent, and it consists of open-domain, fact-based questions with broad semantic categories. Questions may also require a specific order of outcome, or aggregate or filter results. The information conveyed in several languages on the internet. While RDF (Lassila & Swick, 1998) data used to represent the tags in several languages, there is no popular language used in Web documents. People have different native tongues. Using QA systems which can handle several input languages and differ from the language used to communicate information is a more flexible solution. Ambiguity is the phenomenon of different definitions of the same word. It may be textual and syntactic, or functional/semantic and lexical.

The QA system begins the process by reviewing the query as an input, then selecting the appropriate KB in which the relevant answer is then to extract the information from the KB and address the user. Based on KB, modern technology used to store complex structured and unorganized data. In the KB, the fact expressed by a triple as subject, predicate, and objects. The predicate word represents the relationship among the entities. The most popular KBs are Dbpedia (Auer et al., 2007), Yago (Suchanek et al., 2007), Wikipedia (Vökel et al., 2006), Freebase (Bollacker et al., 2008), SPARQL (Cyganiak, 2005) is the standard way to access KB. It is a tedious and challenging process for end-users to access/use due to the complexity of understanding the schema and syntax of SPARQL. The common challenges for developing the QA project are complex queries, where the corresponding SPARQL query contains multiple generic graph patterns and specialized approaches needed to obtain the actual query layout.

The intend of the proposed work is to hide the structure of the model by providing a user-friendly system to the clients with high-performance, and it automatically generates the SPARQL query to retrieve the answer from KB. The primary concept of the approach is a template-based selection that improves the accuracy of an answer in minimal time. These templates are also defined for comparatives and superlatives to cover all the cases. The performance level determines the level of accuracy of the answer given by the system.

The rest of the paper discussed the related work in knowledge-based question answering systems in section 2 and explained the proposed approach and illustration of the process in section 3. Next, it describes the experimental set-up in section 4. Ultimately, the paper concluded in the final section.

2 RELATED WORK

With the growth of semantic mark up to the large-scale data available on the web, there is an arising need for question answering system that can help people utilize the information. Many researchers have investigated in the field of Question Answering Systems in focus with the Semantic Web. To the best of author’s knowledge, here, some of the Question Answering Systems are investigated and examined in this section.

Diefenbach et al. (2020) proposed a system to retrieve answers directly by converting the question into SPARQL. They focus on the semantics of the string to understand the problem. The main advantage of this system is that it is multilingual. Besides, it is working well for both natural language questions and keywords. This system could be easily adaptable to new KBs as well as with querying multiple KBs simultaneously. This QA system proposes a new idea for using semantic words rather than the syntax of the expressions present in the question.
Ricardo et al. (2018) developed a template-based QA system with separate algorithms for listing out the combination of adjacent words and for gathering information such as entities, classes, properties. This system used the QALD-8 dataset Usbeck, Ngomo, Conrads et al., (2018) QALD-8 Dataset, (n.d.) and has many ways to access that includes interface from the GERBIL QA benchmark that provides the answer in the JSON file in the SPARQL query format.

Dubey et al. (2016) proposed a system that helps users to get appropriate answers from RDF KBs for the queries they post. Normalized Query Structure (NQS) facilitates the detection of intended information as output and user-provided input query information and the establishment of a semantic relationship between them. It is also adaptable enough to paraphrase the query. The framework is examined in terms of the syntactic stability of NQS and semantic exactness in standard benchmark data sets and found to be more durable. The issues persist when the system that is sometimes system could not resolve proper relation between input and desire to the correct DBPedia property.

Sen Hu et al. (2017) came up with a systematic concept for extracting answers for the question over the RDF kind of KB repository. They provide a procedural query graph that structurally models the query to reduce the graph to subgraph for matching the problem. The critical step here is to determine and solve the natural language questions ambiguity when the query matchings. By doing so, the disambiguation costs will be saved if there is no matching. Intensive experiments prove that this method not just to increases accuracy, but also significantly accelerates query performance.

Dennis Diefenbach et al. proposed a conceptual solution to overcome the drawbacks of multilingual natural language keywords Diefenbach et al., (2018). This QA system helps the end-users to access the new structured data quickly and efficiently since the existing systems are not capable of adapting to different languages and KBs. They introduced a conceptual method to overcome multiple KBs and language problems. The proposed algorithm is examined and found to work for multiple languages, as well as numerous KBs. The features used helped with the approach of portability that is an added advantage of this proposed system.

H. Jin et al. provides the end-users with an excellent natural language interface and helps them to overcome the complexity of the underlying KB Jin et al., (2019). This QA system allows people who do not have any prior knowledge about the KBs and can get answers for even complex questions. The KBs with triple patterns used for getting candidate subgraphs. The subgraphs and the triple patterns used for the semantic similarity of the answer. To reduce the complexity of this process, the author proposed an extension method based on semantic similarity and identifying the entities and relations while encountering the underlying kb. Then the process is proceeded with creating the query graph. The final process is done with all the preliminary information about the KBs and is followed by tuning the vectors to make them accurate for retrieving answers.

K. Höffner et al. (n.d.) suggested a new technique of QA to multi-dimensional linked information utilizing the RDF Data Cube Vocabulary Höffner et al., (2016), which cannot be interpreted by current methods. They use questions of accessible domain statistical knowledge requirements to evaluate whether these questions vary from others, what extra verbalizations are widely adopted, and how this impacts QA design decisions on statistical data.

Balikas et al. (2015) participated in the competition that consists of semantic indexing but also a QA component on biomedical information. For the QA section, systems are supposed to be combinations, delivering matching triples but also text fragments, but a limited assessment is also feasible. The primary function of forming a description by sorting procedures related to Named Entity Recognition (NER) and Named Entity Disambiguation (NED) and the next function is to incorporate these two moves.

Baudiš et al. (2015) proposed an open-source hybrid solution developed on top of the Apache UIMA framework, part of the Brmson initiative, and influenced by DeepQA. YodaQA enables efficient parallelization and utilization of pre-existing NLP UIMA elements by describing each artifact (question, request answer, passage, applicant response) as a different UIMA CAS. Yoda pipeline has five specific stages: (a) Query Analysis, (b) Response Processing, (c) Response Analysis, (d) Answer Blending and Rating, and (e) Successive Optimizing.
Hakimov et al. (2015) came up with a semantic parsing methodology to QA that achieves high efficiency but depends on a massive volume of training data that is not realistic. At the same time, the scope is broad or unspecified. Mishra et al. (2016) introduce eight classification parameters, such as application context, type of query, the form of data. For each parameter, the various classifications are provided with their benefits, drawbacks, and excellent systems. Park et al. suggested a method to address questions regarding natural language by regular expressions and keyword questions with a Lucene-based index Park et al., (2015). Also, the technique utilizes DBpedia Lehmann et al., (2015) and its triple extraction process for Wikipedia.

SemBioNLQA Sarrouti & El Alaoui, (2020) is a biomedical Question Answering System for extracting answers to biomedical questions from peer-reviewed scientific articles. It uses hand-crafted lexico-syntactic patterns and SVM for classifying the question type, PubMed search engine for document retrieval, BM25 model for passage retrieval and Biopedia synonyms, Term frequency metric for answer extraction. The system can give correct answers for yes/no, factoid and list questions and provides ideal answers for summary questions.

The proposed QA system gives answer responses to questions by building and querying the SPARQL queries in DBPedia Endpoint. This method develops predefined templates for all the input types to prove appropriate answers to the input request. In this approach, the QA system uses annotators, parsers, taggers from Stanford core NLP, DBPedia Spotlight for NER, which results in complexity reduction of the system and providing better performance. The improvement of the system takes place by classifying questions and creating SPARQL template according to its specific type, such as comparative and superlative questions.

### 3 PROPOSED APPROACH

The proposed QA system is developed on the template-based concept by defining SPARQL Query templates that are contoured based on input questions at a predefined location. The QA system consists of the Question Processor and SPARQL Query Builder Grafkin et al., (2016) components. In the question processing phase, the annotations have done on the input to get all necessary details, such as finding relevant entities, question types, classes, and properties. The SPARQL Query builder component uses the collected information for building SPARQL Query by filling gathered information on predefined templates and run it on an endpoint that is provided by DBpedia. The DBpedia endpoint sends the answer with the SPARQL query in JSON format. Figure 1 illustrates the functional flow of the Question Answering System.

#### 3.1 Question Answering System Components

##### 3.1.1 Question Processing

The Question processor plays a vital role in the QA system. Since the SPARQL query is not developed without analyzing the input, this phase can get the necessary information to fetch the appropriate answer and identifies the analyzed relevant entities in the question to find the question type. It helps us to know the kind of response to be retrieved, and Table-1 shows the types of questions and their answers.

The question type corresponds to the question syntactic form. The detection of a question type gives us a clue to determine the different possible expressions of the answer and will also serve to extract the other question features. Question analysis is performed to infer features from questions in order to use them for the extraction of potential answer. These question features would help us to determine the question type with which a list of patterns for extracting the answer will be associated and type of answer required. The factoid questions are related to facts, events, suggestions, and ideas. For example, consider the following questions:
"How do you make a ball?" (Process Question)
"What does extend definition mean, and how would one write a paper on it?". (Formation of two question words)
"The cerebellum is in what part of the body?" (Question word is in the middle).” In which year did India get Independence?” (Question start with preposition)

The above questions represent various types of representation of question posted from the user. In this study, “Wh” questions were performed, and these questions contain unique features, structures, and characteristics that help us to identify and characterize them according to their question types Cheng, (1991). The proposed approach finds the answers for the following questions such as, What, Where, When, Who, How many, How much, Who, When and List questions.

The QA system sends the gathered details to the DBpedia spotlight for deriving named entities. All the found entities are acquired and stored for processing in other components. The Stanford core NLP helps us to find the keywords and derive verbs, adjectives, nouns that support us in finding DBpedia properties and classes. QA system needs to differentiate and remove entities that may also be linking to class to reduce confusion. Then, the system still gets a list of comparatives and superlatives for further elimination of the list of selected templates and find a more appropriate one. These found properties fill the SPARQL query templates. Then, the selection of the SPARQL template takes place according to question type using the SPARQL template given in the following subsection.

| Question Type  | Expected Answer Type           |
|---------------|-------------------------------|
| Where         | Location or Place             |
| When          | Time or Date or Year          |
| Who           | Person                        |
| How           | Number                        |
| Name, List, Show | List or Set or Group      |
| Does, Was, Did, Has, Do | Yes/No, True/False |

Figure 1. Functional flow of Proposed Question Answering System

Table 1. List of Question Types and its Expected Answer Type
3.1.2 **SPARQL Query Builder**

The SPARQL Query builder component has predefined templates for various question types showed in Table 1. QA system then adds the properties, classes, and entities to the selected model found from the question processing component based on the question type. The templates for developing SPARQL queries for different question types are given in Table 2:

**Table 2. Various Question types with corresponding SPARQL templates**

| S.No | Question Types | Templates |
|------|----------------|-----------|
| 1    | “WHERE”        | //uses simple SPARQL template query that accepts properties with the range is dbo:place  
If (list of entities is not empty) {  
  If (size of entity list if equal to 1) {  
    If (the property is not empty) {  
      //uses the available entities and properties with the place range to formulate query  
      //Query is executed and result is added to answer container.}  
  }  
} |
| 2    | “WHAT”         | If (question has superlative) {  
  //uses template for superlative  
}  
Else {  
  //uses simple SPARQL template  
} |
| 3    | “SUPERLATIVE”  | If (question have entity) {  
  // Then we use that to find classes and properties. Then these properties and classes are used to query the superlative.  
}  
Else {  
  //The query is built with the found classes.  
} |
| 4    | “WHERE” with superlative | If (there is a superlative) {  
  //uses superlative template  
}  
Else {  
  //uses simple SPARQL template  
} |
| 5    | “WHEN”         | //Uses simple SPARQL template and sets the filter for dates  
//datatype(?answer)=xsd:date |
| 6    | “WHO”          | If (question has superlative) {  
  //it uses superlative SPARQL template  
}  
Else if (question has most or least) {  
  //it uses determiner template for question type WHO  
}  
Else {  
  //use simple SPARQL template  
} |
| 7    | “WHO” with “Determiner” | If (question contains keyword “most”)  
{  
  //Result is sorted in descending order.  
}  
else if(question contains keyword “least”)  
{  
  //Result is sorted in Ascending order.  
}  
else  
{  
  //It uses simple SPARQL template that accepts properties with the range “PERSON”  
}  
//sets the order to Ascending or Descending |
### 3.2 Illustration

Let us see how the proposed QA system work for “Question: Which Indian Company has the most employees?”

The process starts by adding an annotator from Stanford Core NLP to the question string. Then the question string will split into tokens (that is words) by using PTB tokenizer. The PTB tokenizer is a class provided by Stanford core NLP for the tokenizing process.

Which / Indian / Company / has / the / most / employees / ?

Now, the Parts of Speech (POS) tagging on the tokenized split words takes place by adding annotator POS by Maxent tagger from Stanford NLP.

| S.No | Question Types          | Templates                                                                 |
|------|-------------------------|---------------------------------------------------------------------------|
| 8    | Simple SPARQL query     | If (question has order of “Entity” + “Property” + “Answer type”) {
|      |                         | //builds and sets SPARQL query in the above order.                       |
|      |                         | }                                                                          |
|      |                         | Else {                                                                   |
|      |                         | // Sparql query with ?answer property entity                              |
|      |                         | Else {                                                                   |
|      |                         | //Builds and sets SPARQL query in the other order.                        |
|      |                         | a ?answer rdf:type relation                                               |
|      |                         | }                                                                          |
| 9    | “ASK” Question          | If (Question has 1 Entity) {
|      |                         | //selects query with 1 entity                                             |
|      |                         | }                                                                          |
|      |                         | Else if (Question has 2 entities)                                        |
|      |                         | {//selects query with 2 entities                                          |
|      |                         | }                                                                          |
|      |                         | Else {//It returns false}                                                |
| 10   | “HOW”                   | If (question contains “much”) {
|      |                         | //uses simple SPARQL template                                            |
|      |                         | }                                                                          |
|      |                         | Else if (question contains “many”)                                        |
|      |                         | {//Uses Count(distinct) and builds the query                             |
|      |                         | }                                                                          |
|      |                         | Else {                                                                   |
|      |                         | //uses comparison set to find difference and finally formulating query   |
|      |                         | }                                                                          |
| 11   | “COMPARATIVE”           | If (input have entity) AND (question is not null) AND (check for comparative keywords) {
|      |                         | //It gets URI for the found comparatives using compare function          |
|      |                         | //Then use properties, classes along with URI to query the comparative.   |
|      |                         | }                                                                          |
|      |                         | Else {                                                                   |
|      |                         | //uses simple SPARQL query                                                |
|      |                         | }                                                                          |
| 12   | “LIST”                  | If (there is a comparative) {
|      |                         | //uses comparative template                                             |
|      |                         | }                                                                          |
|      |                         | Else {                                                                   |
|      |                         | //uses simple SPARQL template                                             |
|      |                         | //prints property list of matched properties                              |
Algorithm: Query formulation algorithm

| Input   | Question q, Question type t |
|---------|-----------------------------|
| Output  | SPARQL query                |

BoolQuestion = Arrays.asList("DO", "DID", "HAS", "WAS", "HAVE", "DOES", "WERE", "IS", "ARE", "BE");

AnswerType = Arrays.asList("RESOURCE", "BOOLEAN", "DATE", "NUMBER", "STRING", "LITERAL", "NUMBER");

builder = new SparqlQueryBuilder(q);

result = null;

do

(1) switch(t)

| Case    | Code |
|---------|------|
| "WHO"   | if (q.contains(superlative)) then result = builder.sparqlWho(); break; |
| "HOW"   | result = builder.sparqlHow(); break; |
| "WHERE" | result = builder.sparqlWhere(); break; |
| "WHAT"  | if (q.contains (superlative)) then result = builder.supersparql(); else result = builder.simplesparql(); break; |
| "WHICH" | if (q.contains(superlative)) then result = builder.listSparql(); else result = builder.simplesparql(); break; |
| "WHEN"  | result = builder.simpleSparql(""," FILTER ((datatype(?answer) = xsd:date) || (datatype(?answer) = xsd:gYear))"); |
| default | if (t.equals("LIST") || t.equals("NAME") || t.equals("SHOW") || t.equals("GIVE")) then result = builder.sparqlList(); end if if(BooleanQuestion.contains(t)) then q.questionType = "ASK"; result = builder.simpleASK("",""); end if else result = builder.simpleSparql("",""); end break; |

end

while((result === null || result.isEmpty()) && q.entityList.size() > 1 && builder.getEntityIndex() < q.entityList.size());

container.setAnswers(result);
container.setSparqlQuery(builder getLastUsedQuery());
return container;

end

builder.incrementIndex();

end
Which / Indian / Company / has / the / most / employees / ?
WDT JJ NNP VRB DT RBS NNS

The dependency parser helps in examining the grammatical form of the question and develop the relationship between the lemma and words. Then, the QA system uses Neural Network Dependency Parser for gathering typed dependencies.

Lemmatization:
Which / Indian / Company / has / the / most / employees / ?
Which India Company have the most employee

The question passed to DBpedia Spotlight for identifying entities. The process starts by recognizing phrases that denote reference of DBpedia knowledge. Then the spotted phrases are mapped to resources for selection of candidates. Then, process follows by using the context of spotted phrases for the selection of appropriate candidate. By configuration parameters, annotation may be tailored by users to their particular needs.

Named Entity Recognition:
Which / Indian / Company / has / the / most / employees / ?
Country
Dependency Parser:

The root is the root words that denote grammatical relations. The advmod of a word is an adverb modifier that lists adverb headed expression or adverb to modify the meaning of the word. The dep denotes dependency, where parser cannot establish a more detailed relationship of dependence between two terms. The nsubj is a noun phrase that denotes the subject. The punct means punctuation. The det means determiner that denotes the interlinking of singular proper noun headwords and its determiner. Figure 2 illustrates the output of the Dependency Parser for the given example question.

Now, the system gathers necessary information such as nouns, verbs, adjectives. The IndexDBO class from the library of QA.annotator is used with the extracted nouns (that are classes), verbs, and adjectives (denotes properties) to DBPedia ontology to find the classes and properties.

Figure 2. Dependency Parser Result
The process follows by listing the adjectives, nouns, and classes along with abbreviation and its expansion.

**Adverb:** most  
**Nouns:** Indian, Company, Employee  
**Classes:** [https://dbpedia.org/ontology/Company](https://dbpedia.org/ontology/Company),  
[https://dbpedia.org/ontology/location](https://dbpedia.org/ontology/location),  
[https://dbpedia.org/ontology/numberOfEmployees](https://dbpedia.org/ontology/numberOfEmployees)  
**Entities:** [Indian (URI: https://dbpedia.org/resource/India)](https://dbpedia.org/resource/India)

The question type is WHICH, which assists in selecting the appropriate SPARQL template from the predefined templates with the help of classes, entities, and properties. Then, the system retrieves answers by processing the SPARQL query, and send the response in the JSON format.

```
SELECT DISTINCT ?uri WHERE { ?uri a <https://dbpedia.org/ontology/Company> . ?uri <https://dbpedia.org/ontology/location> <https://dbpedia.org/resource/India> . ?uri <https://dbpedia.org/ontology/numberOfEmployees> ?n . } ORDER BY DESC(?n) OFFSET 0 LIMIT 1

Question: [{string=Which Indian Company has the most employees?, language=en}]
Answer: URI [https://dbpedia.org/resource/Indian_Railways]
```

The inbuilt templates will have SPARQL queries for various question types like where, what, when, who, how. When the SPARQL query builder module finds any comparatives or superlatives, it selects the template accordingly. Table 3 shows the sample enum list for comparing adjectives. The next step is ranking the properties that will return the query with most triples on DBpedia. The answer is made possible in a SPARQL format.

### 4 RESULTS AND DISCUSSION

QALD is a series of evaluation campaigns on question answering over linked data. QALD Contest intends to mediate between the clients and responding to his / her demands of content in human language, as well as RDF information, for an up-to-date review and comparison QALD-8 Dataset.

| Keyword | URI | Order |
|---------|-----|-------|
| High    | dbo:elevation | NIL   |
| Higher  | dbo:elevation | DESC  |
| Highest | dbo:elevation | DESC  |
| Tall    | dbo:height    | NIL   |
| Taller  | dbo:height    | DESC  |
| Tallest | dbo:height    | DESC  |
| Large   | dbo:areaTotal | NIL   |
| Larger  | dbo:areaTotal | DESC  |
| Largest | dbo:areaTotal | DESC  |
It is essential to follow methods that can tackle not only the unique aspects of organized data but also the processing of data across various organized and unorganized knowledge data sources and combined them into a single outcome. Training data consist of 250 questionnaires is gathered and organized from earlier contests. The questions are available in languages like English, Italian, Spanish, Hindi, French, Dutch, Romanian, Farsi and German. The underlying RDF dataset will be DBpedia 2016-10. The questions are general, open-domain and factoid in nature. The questions vary with respect to their complexity, including questions with counts, superlatives, comparatives, and temporal aggregators. Each question is annotated with a manually specified SPARQL query and answers. In order to provide an unbiased test set, real time questions and query logs which express the information needs are compiled and manually curated. The test dataset contains 50 to 100 manually compiled similar questions. The additional questions are from original, real-life inquiry records, and survey files since they have high-grade criteria.

The proposed QA system gives answer responses to user questions by building and querying the SPARQL queries in DBpedia Endpoint. In this approach, the proposed system uses annotators, parsers, taggers from Stanford core NLP, DBpedia Spotlight for Named Entity Recognition, which results in complexity reduction of system and providing better performance. The question is split into tokens by using PTB Tokenizer, a default tokenizer of Stanford Core NLP, for the tokenization process. Then, POS tagging is done on question string using Maximum Entropy tagger. Dependency parser is used for analyzing and knowing the grammatical question structure and get a clear understanding of the relationship with words. After parsing, we collect the information which will be used later for finding the answer. The DBpedia spotlight can find the named entities available to understand its nature. The model classifies the question types and finds an appropriate SPARQL query template. The gathered details such as properties, entities and classes will now fill the selected SPARQL query templates, and Apache Jena helps us to find the possible property set and then the DBpedia endpoint is queried. The data from DBpedia is indexed using the Lucene Library. The executed query results are then ranked with their properties and finds the most triples with DBpedia. After the ranking process, the query at the top of the list will mostly find the correct answer for the user question. Finally, the user request will be answered using a JSON response object.
Figure 3 shows the total number of questions under different question types provided in the test dataset and the count of the answered and unanswered questions. In answering the “WHAT” question type, the count of answered questions is comparatively more than the unanswered questions. Some questions remain unanswered because they come under the SPARQL template categories that are not predefined by the proposed approach. In Table 4, the value is more for unanswered questions for which SPARQL not generated, which shows the need for developing templates for such categories. The “HOW” question type shows a similar count as the “WHAT” question type as it answered most of the questions. The QA system needs more SPARQL templates to ensure that every input question has its matching category of templates to overcome these issues. The results show similar value for “WHEN” and “WHERE” question types. The system shows poor results for “WHO” question types as the problem seems to arise from gathering information regarding the question staring from tagging, finding entities till selecting the appropriate templates. Again, the rectification of these problems can occur by finding and filling up all the unchecked cases. The WHICH question type has zero unanswered questions that mean the proposed method for answering the question using SPARQL templates work well for WHICH question type. The questions unanswered even when the SPARQL template is generated may be due to the error that might have occurred while mapping properties and entities. The framework needs some more templates for tackling the question that is not falling under the developed templates. Minor improvements in these processes will improve results for all question types.

The QALD8 data emphasize questions that include comparative, superlative, and temporal aggregates. So, the QA system needs predefined SPARQL templates to address those kinds of queries. The program would likely give poor outcomes on specific question sets. It could be resolve by making small improvements that may boost the choice of models for Boolean and list queries. QA framework establishes the basis for addressing a variety of queries, and it can quickly expand to enhancement the result further. The problem could have occurred when the approach misunderstood the meaning of the question. So, recognizing the questions and tracking them to classes and entities helps us to characterize the questions that correspond to its suitable SPARQL query.

The QALD8 challenge has an evaluation tool called GERBIL QA. It is a generic benchmarking framework for Linked Data (formerly used by BAT-based object annotation systems), which provides a web-based, easy-to-use interface for agile annotator analysis utilizing various databases and standardized measurement approaches. To connect a device in this framework, the end-user needs to supply application with a URL to the REST interface that satisfies a standard. The tool integration and benchmarking against specified datasets takes place automatically. The Apache Jena, along with SPARQL queries, benefits us by giving some proper meaning with the help of possible property set, and then the DBpedia endpoint is queried. Finally, the request will be answered by using the gerbil wrapper class and complying with a JSON response object. This tool is open sourced anyone can make use of it. The answer set is the input to the GERBIL QA tool for the process of evaluation.

### Table 4. Unanswered Question Analysis

| Question Type | Total Number of Unanswered Questions | Number of Unanswered Questions After constructing SPARQL template | Number of Unanswered Questions Out of Scope |
|---------------|-------------------------------------|---------------------------------------------------------------|-----------------------------------------------|
| What          | 35                                  | 15                                                         | 20                                             |
| How           | 20                                  | 10                                                         | 10                                             |
| Who           | 45                                  | 10                                                         | 35                                             |
| When          | 10                                  | 5                                                         | 5                                             |
| Where         | 10                                  | 0                                                         | 10                                             |
| Which         | 0                                   | 0                                                         | 0                                             |
The evaluation takes place with precision and recall. Precision is a measure of quality, and recall is a measure of quantity. The Equations 1, 2, 3 show the formula for calculating precision and recall.

\[ \text{Precision} = \frac{\text{Number of Correct Answers retrieved}}{\text{Total number of answer retrieved}} \] (1)

\[ \text{Recall} = \frac{\text{Number of Correct Answers retrieved}}{\text{Total no. of Gold Standard Answers}} \] (2)

\[ \text{F -Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \] (3)

The F-score calculation takes place using precision and recall value by Equations 1, 2, and 3. The F-score is the harmonic mean of precision and recall from the experiment. At the value of 1, the F score hits the best value, indicating optimal precision and recall.

The experiment is taking place with two datasets, QALD 8 training dataset, and QALD8 test dataset. When evaluating, QA system performance across different datasets with macro-averaged metrics. The formula for calculating macro averaged metrics given in Equations 4, 5, and 6.

\[ \text{Macro-Precision} = \frac{\text{Precision}_1 + \text{Precision}_2}{2} \] (4)

\[ \text{Macro-Recall} = \frac{\text{Recall}_1 + \text{Recall}_2}{2} \] (5)

\[ \text{Macro-F-Score} = 2 \times \frac{\text{Macro-Precision} \times \text{Macro-Recall}}{\text{Macro-Precision} + \text{Macro-Recall}} \] (6)

If the dataset size is variable, the process uses micro-averaged metrics. The formula for calculating micro averaged metrics as shown in Equations 7, 8, 9,

\[ \text{Micro-Precision} = \frac{\text{True Positives}_1 + \text{True Positives}_2}{\text{True Positives}_1 + \text{False Positives}_2 + \text{True Positives}_2 + \text{False Positives}_2} \] (7)
Micro-Recall = \frac{\text{True Positives}_1 + \text{True Positives}_2}{\text{True Positives}_1 + \text{False Negatives}_1 + \text{True Positives}_2 + \text{False Negatives}_2} \quad (8)

\text{Micro-F-Score} = 2 \times \frac{\text{Micro-Precision} \times \text{Micro-Recall}}{\text{Micro-Precision} + \text{Micro-Recall}} \quad (9)

The evaluation process has to follow the conditions below,

Condition-1. The precision, recall, and F-score are fixed to 1, having an empty answer set, and system answers with the no answer.
Condition-2. The precision, recall, and F-score are fixed to 0, having an empty answer set, but system responds with an answer.
Condition-3. The precision, recall, and F-score have fixed to 0, with an answer set, but the system does not answer.

The evaluation process for the proposed approach takes place using macro and micro F-score. It means that for calculating F-score, precision, and recall, the evaluation process uses true positives, true negatives, false positives, and false negatives by summarizing it. Figure 4, shows the values of the QA system with the QALD8 test and training datasets. This study focused only on the Macro F1 QALD parameter for the final evaluation and the metric incorporates all the above additional qualitative details shown in Figure 4. For example, when the dataset of golden answers are not empty and the proposed system responds with a null answer set, then it fixes the F-score and recall to 0, and precision to 1. The micro F1 score is 0.42, which shows the percentage of the questions correctly answered. Finally, the QA system produces the F1 score as 0.52, which presents the effectiveness of the proposed QA system. The results in Table 5 show that the proposed system outperforms the other competing systems.
5 CONCLUSION

This paper presents a Template-based approach for answering factoid questions over a large Knowledgebase DBpedia. The proposed system successfully answers to the user’s questions by converting the natural language question into a formal query language SPARQL and querying the DBpedia Knowledgebase at its endpoint. We also investigate various question types and develop separate SPARQL templates for each of them. While keeping the model simple by the use of templates, the proposed approach is able to achieve competitive results. We found that the choice of POS tagger, Parser and Named Entity Recognizer has a big impact on identifying the corresponding entities and properties in the DBpedia Knowledge Base. The system experiments with comparative and superlative questions and it is able to answer those which makes the system reliable. As the proposed model does not support all the question types, the authors believe it to be easily portable if templates for other questions are added to the system.

In Future work, creation of all kinds of templates which would support any real time questions are to be created and added to the existing model to improve its performance. Exploitation of external lexical resources and Disambiguation mechanisms would help the system to give support to the real time environment of Question Answering.

FUNDING AGENCY

Publisher has waived the Open Access publishing fee.
REFERENCES

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In The semantic web (pp. 722–735). Springer. doi:10.1007/978-3-540-76298-0_52

Balikas, G., Kosmopoulos, A., Krithara, A., Palouras, G., & Kakadiaris, I. A. (2015). Results of the BioASQ tasks of the Question Answering lab at CLEF 2015. Conference and Labs of the Evaluation forum, 1391.

Baudiš, P., & Šedivy, J. (2015). Modeling of the Question Answering task in the YodaQA system. In Experimental IR Meets Multilinguality, Multimodality, and Interaction - 6th International Conference of the CLEF Association. Springer.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008, June). Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 1247-1250). doi:10.1145/1376616.1376746

Cabrio, E., Cojan, J., Gandon, F., & Hallili, A. (2013). Querying multilingual DBpedia with QAKis. In Extended Semantic Web Conference May 26 (pp. 194–198). Springer.

Cheng, L. L. S. (1991). On the typology of wh-questions [Dissertation]. Department of Linguistics and Philosophy, Massachusetts Institute of Technology.

Cyganiak, R. (2005). A relational algebra for SPARQL. Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170, 35, 9.

Diefenbach, D., Both, A., Singh, K., & Maret, P. (2020). Towards a question answering system over the semantic web. Semantic Web. Preprint.

Diefenbach, D., Singh, K., & Maret, P. (2018). WDAqua-core1: a question answering service for RDF knowledge bases. In Companion Proceedings of the The Web Conference International World Wide Web Conferences Steering Conference (pp. 1087-1091). doi:10.1145/3184558.3191541

Dubey, M., Dasgupta, S., Sharma, A., Höffner, K., & Lehmann, J. (2016). Asknow: A framework for natural language query formalization in SPARQL. In European Semantic Web Conference (pp. 300-316). Springer, Cham. doi:10.1007/978-3-319-34129-3_19

Grafkin, P., Mironov, M., Fellmann, M., Lantow, B., Sandkuhl, K., & Smirnov, A. V. (2016). SPARQL Query Builders: Overview and Comparison. Academic Press.

Hakimov, G. S., Unger, C., Walter, S., & Cimiano, P. (2015). Applying semantic parsing to QA over Linked Data: Addressing the lexical gap. Natural Language Processing and Information Systems: 20th International Conference on Applications of Natural Language to Information Systems, 103–109.

Höffner, K., Lehmann, J., & Usbeck, R. (2016). CubeQA—question answering on RDF data cubes. In International Semantic Web Conference (pp. 325-340). Springer.

Hu, S., Zou, L., Yu, J.X., Wang, H., & Zhao, D. (2017). Answering natural language questions by subgraph matching over knowledge graphs. IEEE Transactions on Knowledge and Data Engineering, 30(5), 824-37.

Jin, H., Luo, Y., Gao, C., Tang, X., & Yuan, P. (2019). ComQA: Question Answering Over Knowledge Base via Semantic Matching. IEEE Access, 7, 75235-46.

Lassila, O., & Swick, R. R. (1998). Resource description framework (RDF) model and syntax specification. http://www. w3. org/TR/PR-rdf-syntax

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Klee, P., Auer, S., & Bizer, C. (2015). DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web, 6(2), 167–195. doi:10.3233/SW-140134

Mishra, A., & Jain, S.K. (2016). A survey on question answering systems with classification. Journal of King Saud University-Computer and Information Sciences, 28(3), 345-61.

Park, Kwon, Kim, Han, Shim, & Lee. (2015). Question Answering system using multiple information sources and open type answer merge. The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 111–115.
QALD-8 Dataset. (n.d.). https://github.com/ag-sc/QALD/tree/master/8/data

RDF Data Cube Vocabulary. (n.d.). http://www.w3.org/TR/vocab-data-cube/

Sarrouti, M., & El Alaoui, S.O. (2020). SemBioNLQA: A semantic biomedical question answering system for retrieving exact and ideal answers to natural language questions. *Artificial Intelligence in Medicine, 102*, 101767.

Suchanek, F. M., Kasneci, G., & Weikum, G. (2007, May). Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web* (pp. 697-706). doi:10.1145/1242572.1242667

Usbeck, R., Gusmita, R. H., Ngomo, A. N., & Saleem, M. (2018). 9th challenge on question answering over linked data (QALD-9). *International Semantic Web Conference*, 58–64.

Usbeck, R., Ngomo, A.C., Conrads, F., Röder, M., & Napolitano, G. (2018). 8th challenge on question answering over linked data (QALD-8). *Language, 7*(1).

Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., & Studer, R. (2006, May). Semantic Wikipedia. In *Proceedings of the 15th international conference on World Wide Web* (pp. 585-594). doi:10.1145/1135777.1135863

Voorhees, E. M. (2001). The TREC question answering track. *Natural Language Engineering, 7*(4), 361–378. doi:10.1017/S1351324901002789

---

Aarthi D. is pursuing her research in School of computer science and engineering, Vellore Institute of Technology, Chennai. She received a Bachelor’s degree in Information Technology at Anna University and a Master’s degree in computer science and engineering at Anna University of Technology. Her research interests in the areas of natural language processing, semantic web, and data mining.

Viswanathan V. is a professor in the School of Computer Science and Engineering at Vellore Institute of Technology, Chennai. He completed his Doctoral degree from Anna University, Chennai, India, by contributing his ideas to the field of Semantic Web Technologies and Social media marketing. He has a teaching experience of over 20 years in the field of Computer Science. His research interests include data mining, semantic web, and social network analysis. He has authored articles in semantic web technologies for renowned publications.