IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 22, 2020, accepted March 9, 2021, date of publication March 22, 2021, date of current version March 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3067942

Question Answering System Over Semantic Web

AARTHI DHANDAPANI AND VISWANATHAN VADIVEL
School of Computer Science and Engineering, Vellore Institute of Technology, Chennai 600127, India
Corresponding author: Aarthi Dhandapani (d.aarthi2013@vit.ac.in)

ABSTRACT The Semantic Web contains a large amount of data in the form of knowledge bases. 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. Although many systems have been developed over the years, it remains a challenge that most systems yet to require improvements to increase the accuracy for correct interpretation of the question and provide an answer. Many Question Answering systems convert the questions into triples which are mapped to the Knowledge base from which 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 which classifies the question types and finds appropriate SPARQL query templates 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 present good performance and can help users to find answers to their questions.

INDEX TERMS Question answering, natural language processing, knowledge base, DBpedia, SPARQL.

I. 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 helps people to retrieve data from the database they want. It can be useful in preserving their private Knowledge Bases(KB) within organizations. They can trust the system to access data sources, other than depending on additional services. The Question Answering system can enact any data source, and can function better as 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, where the origins of data is from various domains. Thus, the scope of QA is enormous and widely acceptable. In Real-time application, the Text Retrieval Conference (TREC) [1] was the first large scale evaluation of being 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 be aggregated or to filter results. The information is conveyed in several languages on the internet. While RDF [2]

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

data is used to represent the tags in several languages, there is no popular language used in Web documents. People have different native tongues but using QA systems can handle several input languages that differ from the language used to communicate information which is a more flexible solution. Ambiguity is the phenomenon of different definitions of the same word, It may be textual, syntactic, functional, semantic or 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 is used to store complex structured and unorganized data. In the KB, the fact is expressed by a triple as subject, predicate, and object. (The predicate word represents the relationship among the entities). The most popular KBs are DBpedia [3], Yago [4], Wikipedia [5], Freebase [6], SPARQL [7] 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 intent 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 paper is organized as follows. Section II discusses the existing literature on Knowledge-based Question Answering Systems. Section III discusses the proposed system and illustration of the process. In Section IV, the experimental findings and the system performances are discussed. The study is summed up in Section V.

II. 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 systems that can help people utilize the information. Many researchers have investigated the field of Question Answering Systems in focus with the Semantic Web. To the best of the authors' knowledge, some of the Question Answering Systems are investigated and examined in this section.

Diefenbach *et al.* [8] 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.

Ngomo [9] 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 [10], [11] 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.* [12] 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 wherever the system could not resolve the proper relation between input and direct to the correct DBpedia property.

Hu *et al.* [13] 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 a subgraph for matching the problem. The critical step here is to determine and solve the natural language question's ambiguity when the query matches. By doing so, the disambiguation costs will be saved if there is no matching similarities. Intensive experiments prove that this method not just 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 [14]. 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 help 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 [15]. 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 are used for getting candidate subgraphs. The subgraphs and the triple patterns are 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.

H öffner *et al.* [16] suggested a new technique of QA to multi-dimensional linked information utilizing the RDF Data Cube Vocabulary [17], 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.* [18] 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. Baudi š and Šedivỳ [19] 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.* [20] 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 and Jain [21] introduces 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 [22]. Also, the technique utilizes DBpedia [23] and its triple extraction process for Wikipedia.

SemBioNLQA [24] 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 Bioportal 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 providing better performance. The improvement of the system takes place by classifying questions and creating a SPARQL template according to its specific type, such as comparative and superlative questions.

III. 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. Figure 1 illustrates the functional flow of the Question Answering System. The QA system consists of two components namely, Question Processor and SPARQL Query Builder. In the question processing, the annotations have been done on the input to get all necessary details, such as finding relevant entities, question types, classes, and properties. The SPARQL Query builder component constructs building SPARQL Query by filling gathered information on predefined templates and run it on an endpoint that is provided by DBpedia. Finally, the DBpedia endpoint sends the answer with the SPARQL query in JSON format.

A. QUESTION ANSWERING SYSTEM COMPONENTS

1) QUESTION PROCESSING

The Question processor plays a vital role in the QA system. To develop a SPARQL query [25], analysing the input question is required. The Question processor identifies the relevant information from the input question to find the question type. It helps us to know the kind of response to be retrieved and Table 1 shows the question types and their expected answer types.

TABLE 1. List of Question Types and its Expected Answer Type.

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

The question type is determined by the syntactic form of the question. The identification of a question type provides us an idea for extracting question features and determining the various potential answers. Analysis of question is performed to infer question features to obtain a relevant 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 the type of answer required. The factoid questions are related to facts, events, suggestions, and ideas. For example, consider the following type of questions which are posted by the user:

"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)

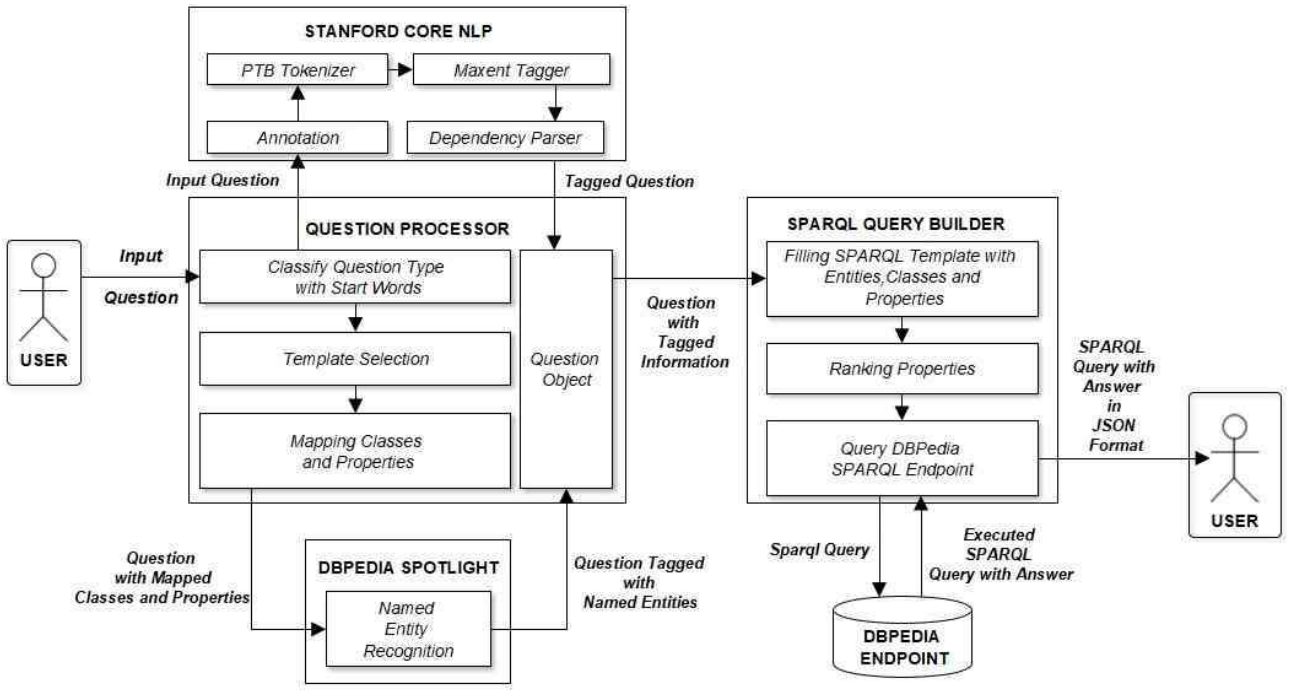
STANFORD CORE NLP
PTB Tokenizer
Maxent Tagger
Annotation
Dependency Parser
Input Question
Tagged Question
QUESTION PROCESSOR
Classify Question Type with Start Words
Template Selection
Mapping Classes and Properties
Question Object
USER
Input Question
Question with Tagged Information
SPARQL QUERY BUILDER
Filling SPARQL Template with Entities,Classes and Properties
Ranking Properties
Query DBPedia SPARQL Endpoint
SPARQL Query with Answer in JSON Format
USER
Question with Mapped Classes and Properties
DBPEDIA SPOTLIGHT
Named Entity Recognition
Question Tagged with Named Entities
Sparql Query
Executed SPARQL Query with Answer
DBPEDIA ENDPOINT

FIGURE 1. Functional flow of Proposed Question Answering System.

"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 a preposition)

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 [26]. The proposed approach finds the answers for the following questions such as, What, Where, How many, How much, Who, When, and List questions. The QA system sends the gathered details to the DBpedia spotlight for identifying the named entities. All the found entities are tagged along with the question and then is sent to the SPARQL query builder component.

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. The following are the patterns that are used to form the templates for the questions. The "+" sign indicates concatenation, "|" sign indicates OR and "*" sign indicates any word.

1. [Where]+[*]+?
2. [What]+[JJS]+[*]+?
3. [Where]+[JJS]+[*]+?
4. [When]+[*]+?
5. [Who]+[JJS]+[*]+?
6. [How]+[much]+[*]+?
7. [How]+[many]+[*]+?
8. [List|Name|Show|Give]+[*]+?

The following subsection explains the selection of the SPARQL template according to question type.

2) SPARQL QUERY BUILDER

The SPARQL Query builder component has predefined templates for various question types that show how to construct a SPARQL query for each question type. The QA system then adds the properties, classes, and entities to the selected template found from the question processing component based on the question type. The templates for developing SPARQL queries for different question types given in Table 2 and Table 3.

The queries are finally executed in the DBpedia endpoint using SPARQL templates designed for various question types, including comparative and superlatives. The query results are ranked according to their properties, and the most number of triples in DBpedia are identified. Following the ranking process, the query at the top of the list will almost always have the correct answer to the user's question. The proposed algorithm for building the SPARQL query from the input question is given in Algorithm 1.

B. 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

TABLE 2. Various Wh-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 a query
		//Query is executed and the result is added to the 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 the keyword "most") {
		//Result is sorted in descending order.
		}
		Else if(question contains keyword "least") {
		//Result is sorted in ascending order.
		}
		Else {
		//It uses a simple SPARQL template that accepts properties with the range "PERSON"
		}
		//sets the order to Ascending or Descending

tokens (that is words) by using the 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.

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

TABLE 3. **Various Question types with corresponding SPARQL templates.**

S.No	Question Types	Templates
1	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 }}
2	ASK	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}
3	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 }
4	COMPARATIVE	If (input have entity) AND (question is not null) AND (check for comparative keywords) { //It gets URI for the found comparatives using the compare function //Then use properties, classes along with URI to query the comparative. } Else { //uses simple SPARQL query }
5	LIST	If (there is a comparative) { //uses comparative template } Else { //uses simple SPARQL template //prints property list of matched properties }

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 (Which) / Indian (India) / Company (Company) / has (have) / the (the) / most (most) / employees (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 the selection of candidates. Then, the process follows by using the context of spotted phrases for the selection of an appropriate candidate. By configuration parameters, the annotation may be tailored by users to their particular needs.

Named Entity Recognition: **Which / Indian (Country) / Company / has / the / most / employees / ?**

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 the

Algorithm 1 Query Formulation Algorithm

```
    Input: Question q, Question type t
    Output: SPARQL Query
1:  BoolQuestion = Arrays.asList (“DO”, “DID”, “HAS”, “WAS”, “HAVE”, “DOES”, “WERE”, “IS”, “ARE”, “BE”);
2:  AnswerType = Arrays.asList (“RESOURCE”, “BOOLEAN”, “DATE”, “NUMBER”, “STRING”, “LITERAL”,
    “NUMBER”);
3:  builder = new SparqlQueryBuilder(q);
4:  result = null;
5:  switch t do
6:      case “WHO”:
7:          if q.contains(superlative) then
8:              result = builder.sparqlWho();
9:              break;
10:     case “HOW”:
11:         result = builder.sparqlHow();
12:         break;
13:     case “WHERE”:
14:         result = builder.sparqlWhere();
15:         break;
16:     case “WHAT”:
17:         if (q.contains(superlative) then
18:             result = builder.supersparql();
19:             result = builder.simplesparql();
20:             break;
21:     case “WHICH”:
22:         if q.contains(superlative) then
23:             result = builder.listSparql();
24:             result = builder.simplesparql();
25:             break;
26:     case “WHEN”:
27:         result = builder.simpleSparql(““,” FILTER ( (datatype(?answer) = xsd:date) || (datatype(?answer) =
    xsd:gYear))”);
        default:
28:     if (t.equals(“LIST”) || t.equals(“NAME”) || t.equals(“SHOW”) || t.equals(“GIVE”)) then
29:         result = builder.sparqlList();
30:     if (BoolQuestion.contains(t)) then
31:         q.questionType = “ASK”;
32:         result = builder.simpleASK(““,””);
        else
33:     result = builder.simpleSparql(““, ””); end
34:     break;
35:     while ( do(result == null ||result.isEmpty()) and (q.entityList != null) and (q.entityList.size()>1) and
    (builder.getEntityIndex()<q.entityList.size()));
36:         container.setAnswers(result);
37:         container.setSparqlQuery(builder.getLastUsedQuery());
38:         return container;
39:     builder.incrementIndex();
```

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.

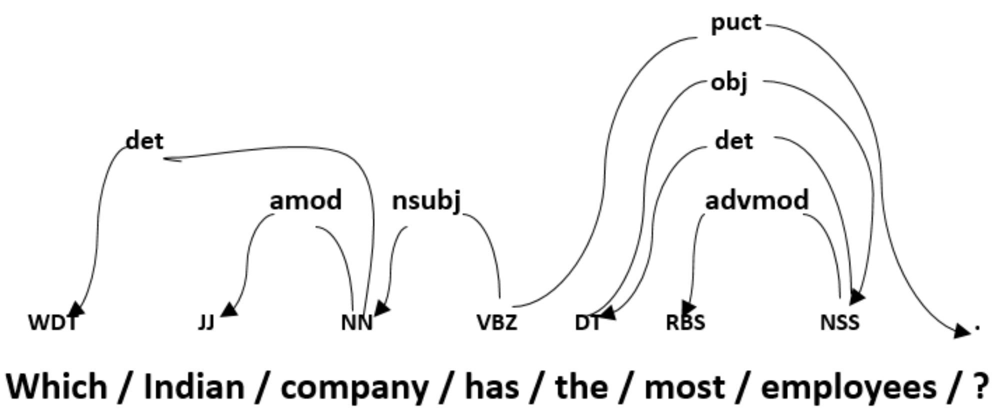

FIGURE 2. Dependency Parser output for the input question.

The process follows by listing the adjectives, nouns, and classes along with abbreviation and its expansion.

Adverb: most

Nouns: Indian, Company, Employee

Classes:

[*http* : *//dbpedia.org/ontology/Company*]
[*http* : *//dbpedia.org/ontology/location*]
[*http* : *//dbpedia.org/ontology/numberOfEmployees*]

Entities:

[Indian (URI: *http* : *//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 &lt; *http* : *//dbpedia.org/ontology/Company* &gt;. ?uri &lt; *http* : *//dbpedia.org/ontology/location* &gt; &lt; *http* : *//dbpedia.org/resource/India* &gt;. ?uri &lt; *http* : *//dbpedia.org/ontology/numberOfEmployees* &gt; ?n ORDER BY DESC(?n) OFFSET 0 LIMIT 1

Question:

[string=Which Indian Company has the most employees?, language=en]

Answer:

URI [*http* : *//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 4 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.

IV. 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 [11]. 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 questions 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 based on their complexity, including questions with superlatives, comparatives, and temporal aggregators. Each question is annotated with a manually specified SPARQL query and answers. To provide an unbiased test set, real-time questions and query logs that 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.

TABLE 4. Sample Enum List for Comparing Adjectives.

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

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,

TABLE 5. Unanswered Question Analysis.

Question type	Total number of questions	Number of unanswered questions	Number of unanswered questions after constructing SPARQL template	Number of unanswered questions out of scope
What	80	35	15	20
How	55	20	10	10
Who	45	45	10	35
When	20	10	5	5
Where	20	10	0	10
Which	15	0	0	0

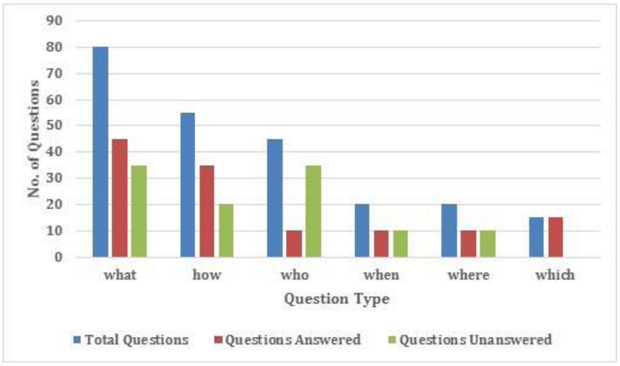

FIGURE 3. Experimental results of various Question Types.

DBpedia Spotlight for Named Entity Recognition, which results in complexity reduction of the system 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 system 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 5, the value is more for unanswered questions for which SPARQL is 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 values for "WHEN" and "WHERE" question types. For eg. the system does not give an answer for the question "Where does Piccadilly start?". The question is about the starting route of the Piccadilly train station, and the answer is "Dover Street". The QA system requires a deep understanding of the keyword "start" to answer this question correctly. In the DBpedia ontology, the QA system should be able to associate the words "start" and "route start". It necessitates the use of disambiguation techniques to determine the appropriate meaning of the word "start" with respect to the context of the question. The system shows poor results for "WHO" question types as the problem seems to arise from gathering information regarding the question starting from tagging, finding entities till selecting the appropriate templates. For eg. the system does not identify the answer for the question "Who played Agent Smith in Matrix?". 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 QALD-8 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 resolved 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 enhance 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 QALD-8 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 the application with a URL to the REST interface that satisfies a standard. The tool integration and benchmarking against specified datasets take place automatically. The Apache Jena, along with SPARQL queries, benefits us by giving some proper meaning with the help of a 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. The evaluation takes place with precision and recall. Precision is a measure of quality, and recall is a measure of quantity. The Equations below show the formula for calculating precision and recall.

$$Precision = \frac{No.\_of\_Correct\_Answers\_retrieved}{Total\_no.\_of\_Answers\_retrieved} \tag{1}$$

$$Recall = \frac{No.\_of\_Correct\_Answers\_retrieved}{Total\_no.\_of\_Gold\_Standard\_Answers} \tag{2}$$

$$Fscore = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

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. At the value of 1, The F score hits the best value, indicating optimal precision and recall.

QALD-8 dataset is used for comparison betweeen proposed method and other existing systems with templates [8], [27] and without predefined template [13] as shown in Table 6. Comparison of various metrics for the proposed system is shown in figure 4. The experiment is evaluated with QA system performance across training and test data with macro-averaged metrics. The formula for calculating macro averaged metrics given in Equations 4, 5, and 6.

$$Macro-Precision = \frac{Precision1 + Precision2}{2} \tag{4}$$

$$Macro-Recall = \frac{Recall1 + Recall2}{2} \tag{5}$$

$$Macro-Fscore = 2*\frac{Macro-precision*Macro-recall}{Macro-precision+Macro-recall} \tag{6}$$

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,

$$Micro-Precision = \frac{TP1 + TP2}{TP1 + FP2 + TP2 + FP2} \tag{7}$$

$$Micro-Recall = \frac{TP1 + TP2}{TP1 + FN1 + TP2 + FN2} \tag{8}$$

$$Micro-Fscore = 2*\frac{Micro-precision*Micro-recall}{Micro-precision+Micro-recall} \tag{9}$$

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 the system responds with an answer. Condition-3: The precision, recall, and F-score have been fixed to 0, with an answer set, but the system does not answer.

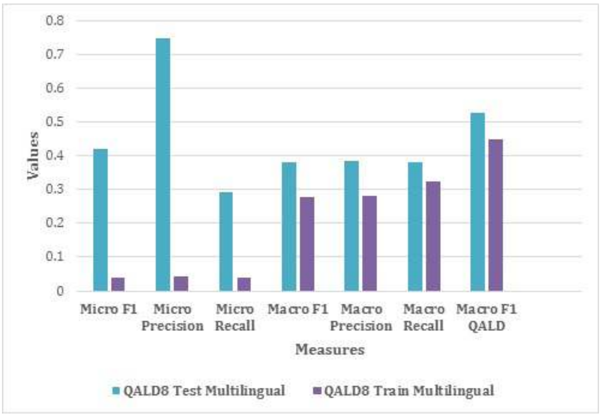

FIGURE 4. Comparison of various metrics for the proposed QA system.

TABLE 6. Comparison of the proposed system over the QALD-8 dataset.

QA System	Precision	Recall	F-measure	Runtime
gAnswer2 [13]	0.39	0.39	0.39	1.92 s
WD-AquaCore [8]	0.39	0.4	0.39	1.72 s
QAKIS [27]	0.06	0.05	0.06	15.41 s
Proposed System	0.75	0.4	0.52	1.8 s

V. 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 system simple by the use of templates, the proposed approach can 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 can 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, the creation of all kinds of templates that would support any real-time questions are to be created and added to the existing model to improve its performance. The exploitation of external lexical resources and Disambiguation mechanisms would help the system to give support to the real-time environment of Question Answering.

REFERENCES

[1] E. M. Voorhees, "The TREC question answering track," *Natural Lang. Eng.*, vol. 7, no. 4, pp. 361–378, Dec. 2001.

[2] O. Lassila. (1999). *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/PR-rdf-syntax

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A nucleus for a Web of open data," in *The Semantic Web*. Berlin, Germany: Springer, Nov. 2007, pp. 722–735.

[4] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proc. 16th Int. Conf. World Wide Web (WWW)*, 2007, pp. 697–706.

[5] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer, "Semantic wikipedia," in *Proc. 15th Int. Conf. World Wide Web*, May 2006, pp. 585–594.

[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2008, pp. 1247–1250.

[7] R. Cyganiak, "A relational algebra for SPARQL," Digit. Media Syst. Lab. HP Laboratories, Bristol, U.K., Tech. Rep. HPL-2005-170, vol. 35, 2005, p. 9.

[8] D. Diefenbach, A. Both, K. Singh, and P. Maret, "Towards a question answering system over the semantic Web," *Semantic Web*, vol. 11, no. 3, pp. 1–19, 2020.

[9] N. Ngomo, "9th challenge on question answering over linked data (QALD-9)," *Language*, vol. 7, no. 1, pp. 58–64, 2018.

[10] R. Usbeck, A.-C. N. Ngomo, F. Conrads, M. Röder, and G. Napolitano, "8th challenge on question answering over linked data (QALD-8)," *Language*, vol. 7, no. 1, pp. 51–57, 2018.

[11] *Qald 8 Dataset*. Accessed: Mar. 19, 2021. [Online]. Available: https://github.com/ag-sc/QALD/tree/master/8/data

[12] M. Dubey, S. Dasgupta, A. Sharma, K. Höffner, and J. Lehmann, "AskNow: A framework for natural language query formalization in SPARQL," in *Proc. Eur. Semantic Web Conf.* Cham, Switzerland: Springer, 2016, pp. 300–316.

[13] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao, "Answering natural language questions by subgraph matching over knowledge graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 5, pp. 824–837, May 2018.

[14] D. Diefenbach, K. Singh, and P. Maret, "WDAqua-core1: A question answering service for RDF knowledge bases," in *Proc. Companion Web Conf. Web Conf. (WWW)*, 2018, pp. 1087–1091.

[15] H. Jin, Y. Luo, C. Gao, X. Tang, and P. Yuan, "ComQA: Question answering over knowledge base via semantic matching," *IEEE Access*, vol. 7, pp. 75235–75246, 2019.

[16] K. Höffner, J. Lehmann, and R. Usbeck, "CubeQA—Question answering on RDF data cubes," in *Proc. Int. Semantic Web Conf.* Cham, Switzerland: Springer, 2016, pp. 325–340.

[17] R. Cyganiak and D. Reynolds. *RDF Data Cube Vocabulary*. Accessed: Mar. 19, 2021. [Online]. Available: http://www.w3.org/TR/vocab-data-cube

[18] G. Balikas, A. Kosmopoulos, A. Krithara, G. Paliouras, and I. Kakadiaris, "Results of the BioASQ tasks of the question answering lab at CLEF 2015," in *Proc. CLEF*, 2015, pp. 1–15.

[19] P. Baudiš and J. Šedivỳ, "Modeling of the question answering task in the YodaQA system," in *Proc. Int. Conf. Cross-Lang. Eval. Forum Eur. Lang.* Cham, Switzerland: Springer, 2015, pp. 222–228.

[20] S. Hakimov, C. Unger, S. Walter, and P. Cimiano, "Applying semantic parsing to question answering over linked data: Addressing the lexical gap," in *Proc. Int. Conf. Appl. Natural Lang. Inf. Syst.* Cham, Switzerland: Springer, 2015, pp. 103–109.

[21] A. Mishra and S. K. Jain, "A survey on question answering systems with classification," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 28, no. 3, pp. 345–361, Jul. 2016.

[22] S. Park, S. Kwon, B. Kim, S. Han, H. Shim, and G. G. Lee, "Question answering system using multiple information source and open type answer merge," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Demonstrations*, 2015, pp. 111–115.

[23] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[24] M. Sarrouti and S. O. E. Alaoui, "SemBioNLQA: A semantic biomedical question answering system for retrieving exact and ideal answers to natural language questions," *Artif. Intell. Med.*, vol. 102, Jan. 2020, Art. no. 101767.

[25] P. Grafkin, M. Mironov, M. Fellmann, B. Lantow, K. Sandkuhl, and A. V. Smirnov, "SPARQL query builders: Overview and comparison," in *Proc. BIR Workshops*, 2016, pp. 255–274.

[26] L. L. S. Cheng, "On the typology of wh-questions," Ph.D. dissertation, Dept. Linguistics Philosophy, Massachusetts Inst. Technol., Cambridge, MA, USA, 1991.

[27] E. Cabrio, J. Cojan, F. Gandon, and A. Hallili, "Querying multilingual DBpedia with QAKiS," in *Proc. Extended Semantic Web Conf.* Berlin, Germany: Springer, 2013, pp. 194–198.

AARTHI DHANDAPANI received the bachelor's degree in information technology from Anna University and the master's degree in computer science and engineering from the Anna University of Technology. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai. Her research interests include the natural language processing, semantic web, and data mining.

VISWANATHAN VADIVEL received the Ph.D. degree from Anna University, Chennai, India, by contributing his ideas to the field of semantic web technologies and social media marketing. He has teaching experience of over 20 years in the field of computer science. He is currently a Professor with the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai. He has authored articles in semantic web technologies for renowned publications. His research interests include data mining, semantic web, and social network analysis.