Core Techniques of Ontology-Based Question Answering Systems: a Survey

Dennis Diefenbach [a], Kamal Singh [a] and Pierre Maret [a]
[a] *Université de Lyon, CNRS UMR 5516 Laboratoire Hubert Curien, F-42023, Saint-Etienne, France*
E-mail: {dennis.diefenbach,kamal.singh,pierre.maret}@univ-st-etienne.fr

Abstract. The Semantic Web contains an enormous amount of information in the form of ontologies. To make this information available, many ontology-based question answering (QA) systems were created in the last years. Building a QA system is difficult since many different challenges have to be solved. To address these challenges, QA systems generally combine techniques from natural language processing, machine learning and Semantic Web. The aim of this survey is to give an overview over the techniques used in current QA systems. We present the techniques used by the QA systems which were evaluated on a popular series of benchmarks: Question Answering over Linked Data (QALD). These techniques are presented in a novel way: techniques that solve the same task are first grouped together and then described. For each technique advantages and disadvantages are discussed. This allows a direct comparison of similar techniques. We conclude that there is a need for a framework to interweave current approaches.

Keywords: Question Answering, QALD, Semantic Web

1. Introduction

Question answering (QA) is a very old research field in computer science. The first QA systems were developed to access data over databases in the late sixties and early seventies. More recent works address the QA problem from an information retrieval perspective: finding the part of text that contains the answer of a question from a set of documents. In the last two decades, thanks to the development of the Semantic Web, a lot of new structured data has become available on the web in the form of ontologies. Nowadays there are ontologies about media, publications, geography, life-science and more[1]. QA over ontologies has emerged in the last two decades to make this information available to the end user. For a more detailed historical overview see [23].
The publication of new ontologies accelerated thanks to the publication by the W3C of the de facto standards RDF[2] and SPARQL[3]. RDF is a format for publishing new ontologies. SPARQL is a powerful query language for RDF, but it requires expert knowledge. The idea behind an ontology-based QA system is to translate a natural question to a SPARQL query that can be used to retrieve the desired information.
Building a QA system is a difficult task since many different techniques are required. It is a combination of techniques from natural language processing (NLP), machine learning and Semantic Web. The aim of this paper is to give an overview of the techniques used by QA systems, and explain their strengths and weaknesses.
Nowadays, there is a large number of QA systems so that analyzing them all would be a huge work. Instead we restrict to QA systems participating to a specific benchmark. This allows a comparison of the overall performance of the QA systems and can give at least a hint of how good are the single techniques used in QA systems. Different benchmarks exist for QA systems.

[1]http://lod-cloud.net

[2]http://www.w3.org/TR/rdf11-mt/
[3]http://www.w3.org/TR/sparql11-query/

One is Free917[7], which consists of 917 questions that can be answered using Freebase. The benchmark contains pairs of questions and semantically equivalent logical forms. The idea behind the benchmark is to use machine learning techniques to learn how to translate a question to an equivalent logical form. This approach has the disadvantage that for each new ontology a new set of questions with the corresponding logical forms has to be created. This requires expert knowledge. Another popular benchmark is WebQuestions[5]. It contains 5810 questions that can also be answered using Freebase. The benchmark contains a set of questions and the corresponding answers. The idea behind this set is to apply machine learning techniques to learn how to construct a query using question-answer pairs.
In this publication, we restrict to QA systems participating to another series of popular benchmarks: Question Answering over Linked Data (QALD). QALD is different from Free917 and WebQuestions in that it does not limit itself to address only a specific task like learning from pairs of questions and logical forms or pairs of questions and answers. This results in a broader spectrum of approaches to solve the question answering problem. Moreover, the QALD benchmark is in line with the principles of the Semantic Web: it focuses on RDF ontologies, it proposes ontologies in different domains assuming that a QA system should be able to query any ontology, it proposes large ontologies like DBpedia such that scalability becomes an issue, it offers questions where the information should be extracted from multiple interlinked ontologies and it proposes multilingual questions assuming that the users can be of different nationalities.
The last survey on QA was published in 2011 [23]. Our survey is different since it concentrates on engines that where created after 2011 and focuses on the techniques used by the different QA systems. Moreover, instead of describing each QA system in a systematic manner, we identified four tasks in the QA process: question analysis, phrase mapping, disambiguation, query construction. For the first three tasks the paper collects the techniques used by the QA systems to solve them. The advantages and disadvantages of each technique are described. How these techniques are combined by the different question answering systems is described in the query construction task. This provides a novel view on the field of ontology-based QA systems.
The paper is organized as follows. In section 2 there is a detailed description of the QALD benchmarks. An overview over the four tasks is given in section 3. In section 4 we describe the techniques used for the question analysis task, in section 5 we discuss the techniques used for the phrase mapping task and in section 6 the techniques for the disambiguation task are provided. In section 7 we describe how the different QA systems combine the techniques to construct a query. Based on the analysis of the paper in section 8 we propose a new strategy to construct QA systems.

2. The QALD benchmark

Benchmarks for ontology-based QA are a new development. Still in 2011 there was no established benchmark that was used to compare different QA systems as described in [23]. Therefore, QA systems were tested on different ontologies in small scale scenarios with ad-hoc tasks. To fill this gap a yearly benchmark was introduced: "Question Answering over Linked Data" (QALD)[4] [21][8][30]. So far five QALD challenges have been launched. Each challenge provided an RDF dataset, a set of natural questions, which should be answered using the provided data, and a set of procedures and metrics for evaluating a system. The challenges included always DBpedia[5] as an RDF dataset besides some others. Since all the participating systems used DBpedia as an underlying dataset and only a few used the other, we concentrate on the results using DBpedia. Table 1 offers an overview of the five challenges, the corresponding DBpedia dataset, and a short description of the questions. For evaluation, three parameters are used: precision, recall, and F-measure. Precision indicates how many of the answers are correct. For a question q it is computed as:

$$\text{Precision(q)} = \frac{\text{number correct system answers for q}}{\text{number system answers for q}}.$$

The second parameter is recall. For each question there is an expected set of correct answers, these are called the gold standard answers. Recall indicates how many of the returned answers are in the gold standard. It is computed as:

$$\text{Recall(q)} = \frac{\text{number correct system answers for q}}{\text{number gold standard answers for q}}.$$

We explain these parameters using the following example. Assume a user asks the question: "Which are the three primary colors?". The gold standard answers would be "green", "red" and "blue". If a system returns

[4]http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php

[5]http://www.dbpedia.org/

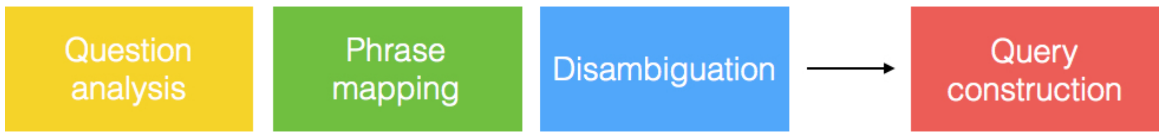


Fig. 1. Tasks in the QA process

as answers "green" and "blue" then the precision is 1 since all answers are correct but the recall is only 2/3 since the answer "blue" is missing.
The precision and recall of each question are used to compute the global precision and recall of a system. Unfortunately the global precision and recall were calculated differently in the different challenges. In QALD-3, QALD-4 they were computed as the average over all questions in the benchmark. In QALD-1, QALD-2, QALD-5 they were computed as the average over all questions to which the system returned an answer, i.e. questions that were not processed were not considered.
The F-measure is the weighted average between the global precision and recall and it is computed as follows:

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Note that the F-measure is near to zero if either precision or recall are near to zero and is equal to one only if both precision and recall are one. Another parameter which is not considered in the QALD challenges, although important for the user, is performance. This is generally indicated as the average time taken by the QA system for answering the questions.

Table 2 contains a list of all systems participating to the QALD challenges and their respective scores. Moreover, many other QA systems not participating to the challenges were evaluated under the same test conditions. The table gives a reference to the publications containing the results of the evaluation. This allows a good comparison between many QA engines implementing different techniques and their impact on precision and recall. For the systems where we found the performance we indicate it. Note that some challenges propose questions in different languages but all participating systems were evaluated only on the English version.
Some systems were excluded in this comparison. This includes the engine Alexandria [32] where the questions were translated from German to English and squall2sparql [15] where the questions were translated to a controlled natural language. The engine Scalewelis [19] was also excluded since it does not take as input a natural language question, but is more like a semantic web browser. SWIP [26] was excluded since it uses templates adapted to a specific ontology so that it is not an open-domain QA system. YodaQA [3] was excluded since it mainly uses information retrieval techniques. Finally, APAQ and RO_FII were excluded since no corresponding publication could be found.

3. The question answering process

To classify the techniques used in QA systems we divide the question answering process in four tasks: question analysis, phrase mapping, disambiguation and query construction (see Figure 1).
In the following we briefly describe each of these steps using a simple example. Assume the user inputs the question

"What is the population of Europe?"

and that DBpedia is the queried ontology.
In the question analysis task, we include all techniques that use purely syntactic features to extract information about a given question. These include for example determining the type of the question, it is a "What" question, identifying named entities, like "Europe", identifying relations, entities and classes, like the relation "is the population of", and their dependencies, like that there is a dependency between "is the population of" and "Europe".
In the phrase mapping task one starts with a phrase s and tries to identify resources that with high probability correspond to s. For example for the phrase "Europe" possible resources in DBpedia are: dbr:Europe _(band)[6] (that refers to a Band called Europe), dbr: Europe (that refers to Europe as a continent) and dbr:Europe _(dinghy) (a particular type of boat).
In the disambiguation task we include techniques that are used to determine which of the resources identified during the phrase mapping task are the right ones. In

[6] @prefix dbr: <http://dbpedia.org/resource/>

Table 1
QALD Challenges

Challenge	Dataset	Questions
QALD-1	English DBpedia 3.6	50 questions in English
QALD-2	English DBpedia 3.7	100 questions in English
QALD-3 Task 1	English DBpedia 3.8 Spanish DBpedia	100 questions in English 100 in Spanish
QALD-4 Task 1	English DBpedia 3.9 with multilingual labels	200 multilingual questions in 7 languages (English, Spanish, German, Italian, French, Dutch, and Romanian)
QALD-5 multilingual questions	DBpedia 2014 dataset for English	300 multilingual questions in 7 languages (English, German, Spanish, Italian, French, Dutch, and Romanian)

Table 2
QALD Benchmark results

Engine	Total	Processed	Right	Partially	Precision	Recall	F-measure	Performance	Reference
QALD-1									
TBSL	50	34	19	2	0.61	0.63	0.62	-	[29]
FREyA	50	43	27	10	0.63	0.54	0.58	36 s	[21]
PowerAqua	50	46	24	13	0.52	0.48	0.50	20 s	[21]
QALD-2									
BELA	100	31	17	5	0.62	0.73	0.67	-	[31]
SemSeK	100	80	32	7	0.44	0.48	0.46	-	[21]
MHE	100	97	30	12	0.36	0.40	0.38	-	[21]
QAKiS	100	35	11	4	0.39	0.37	0.38	-	[21]
QALD-3									
gAnswer	100	76	32	11	0.40	0.40	0.40	$\approx$ 1 s	[39]
CASIA	99	52	29	8	0.35	0.36	0.36	-	[8]
RTV	99	55	30	4	0.32	0.34	0.33	-	[8]
Intui2	99	99	28	4	0.32	0.32	0.32	-	[8]
SINA	100	32	32	0	0.32	0.32	0.32	-	[28]
DEANNA	100	27	21	0	0.21	0.21	0.21	$\approx$ 1-50 s	[39]
QALD-4									
Xser	50	40	34	6	0.72	0.71	0.72	-	[30]
gAnswer	50	25	16	4	0.37	0.37	0.37	0.973 s	[30]
CASIA	50	26	15	4	0.32	0.40	0.36	-	[30]
Intui3	50	33	10	4	0.23	0.25	0.24	-	[30]
ISOFT	50	28	10	3	0.21	0.26	0.23	-	[30]
QALD-5									
Xser	50	42	26	7	0.74	0.72	0.73	-	[1]
QAnswer	50	37	9	4	0.46	0.35	0.40	-	[1]
SemGraphQA	50	31	7	3	0.31	0.32	0.31	-	[1]
YodaQA	50	33	8	2	0.28	0.25	0.26	-	[1]

the above example, "Europe" cannot refer to a band or a boat since it does not make sense to speak about their population. Therefore, these two resources can be excluded.
The query construction task describes how the techniques in the previous steps are combined to construct a SPARQL query that can be send to an endpoint. In the example above, the generated SPARQL query would be:

```
Select ?p where {
      dbr:Europe dbp:populationTotal ?p
}
```

Note that QA systems generally do not follow such a strict division in four task. But we think that every QA system can be decomposed in these four tasks. Moreover, also if the order of the tasks is generally followed it is not always the case.
The following three sections describe the techniques that are used by QA systems to solve the first three tasks. In the section about query construction we describe how the techniques are combined to produce the final SPARQL query. We focus on techniques that can be adapted to any ontology, i.e. techniques that do not use some DBpedia/Wikipedia specific features. Moreover, we will not address techniques to deal with multiple ontologies. Such techniques are for example presented in PowerAqua[20].

4. Analyzing the question

In this step the question of the user is analyzed based on purely syntactic features. QA systems use syntactic features to deduce for example the right segmentation of the question, determine which phrase corresponds to an instance (subject or object), property or class, the dependency between the different phrases and even identify templates for possible SPARQL queries. We classify techniques for question analysis in techniques for recognizing named entities, techniques that use Part-of-speech (POS) tagging, techniques that use parsers, and techniques that use parsers with a light semantic representation.

4.1. Recognizing named entities

An important task in QA is to segment the question, i.e. identify contiguous spans of tokens that refer to a resource. As an example consider the question: "Who directed One Day in Europe?". The span of tokens "One Day in Europe" refers to a film. QA systems use different strategies to solve this problem.

NER tools from NLP
One approach is to use NER tools used in natural language processing. Unfortunately, NER generally are adapted to a specific domain. It was observed in [18] that the Stanford NER tool[7] could recognize only 51.5% of the named entities in the QALD-3 training set.

N-gram strategy
A common strategy is to try to map n-grams (groups of n words) in the question to entities in the underling ontology. If at least one resource is found, then the corresponding n-gram is considered as a possible named entity (NE). This is used for example in SINA[28] and CASIA[18]. This has the advantage that each NE in the ontology can be recognized.

Other tools
Some engines use some other tools to recognize NE. These are DBpedia Spotlight[11] and AIDA[38]. These tools do not only recognize NE, but also find the corresponding resources in the underlying ontology. So they perform many steps at once: identifying contiguous span of tokens that refer to some entity, find possible resources that can correspond to it and disambiguate between them. DBpedia Spotlight and AIDA both use the link structure of Wikipedia and can therefore not be used for general ontologies. DBpedia Spotlight uses a specific training set. It is created by assigning to each anchor text in Wikipedia the resource corresponding to the page it links to. AIDA uses the number of shared incoming links of two wikipedia articles to evaluate if they appear in the same context, i.e. if two entities appear often together they should appear often on the same Wikipedia page.

4.2. POS tagging

Part-of-speech (POS) tagging is the process of assigning the part-of-speech tag like noun, verb and adjective to each word of a sentence. For example, the Stanford POS tagger[8] returns the following POS tags for the question "When was the European Union founded?":

[7]http://nlp.stanford.edu/software/CRF-NER.shtml
[8]http://nlp.stanford.edu/software/tagger.shtml

WRB	VBD	DT	NNP	NNP	VBN	.
When	was	the	European	Union	founded	?

where WRB stands for Wh-adverb, VBD for past tense verb, DT for determiner, NNP for proper singular noun and VBN for past participle verb. Note that it was reported that questions often lead to POS tagging errors [29]. For example, in the question "Which films did Leonardo di Caprio star in?" both the Stanford POS tagger and the Apache openNLP POS tagger[9], tag the word "star" as a noun instead of a verb. It was suggested therefore to train a POS tagger with a sufficient amount of questions in the training set.
POS tags are used mainly to identify which phrases correspond to instances (subjects or objects), to properties, to classes and which phrases do not contain relevant information. For example nouns generally refer to instances and classes (like "European") and verbs to properties (like "founded" above). Moreover, determiners generally do not contain any relevant information (like "the").
The general strategy using POS tags is to identify some reliable POS tags expressions to recognize entities, relations and classes. These expressions can then be easily identified using regular expressions over the POS tags. Now we describe how the POS tag expressions are created: either by hand or using machine learning techniques.

Handmade rules
Some engines rely on hand written regular expressions. This is the case for PowerAqua [20][22]. PowerAqua uses regular expressions also to groups the identified expressions into triples of phrases. To do this PowerAqua relies on an extensive list of question templates, i.e. depending on the type of the question and its structure the phrases are mapped to triples.
Also DENNA uses regular expressions since it relies on ReVerb[14]. ReVerb is an information extraction tool designed to scan big text corpora and extract binary relations from them to be inserted into an ontology. The main point of ReVerb is the observation that most of the expressions corresponding to properties satisfy the following regular expression over POS tags:

V | VP | VW*P
V = verb particle? adverb?
W = (noun | adjective | adverb | pronoun | determiner)
P = (preposition | particle | inf. marker)

This expression matches "was", "founded", but also expressions like "was founded by" and "is the founding year of". It was shown that in a random sample of Web sentences 85% of the binary relations could be identified this way [14]. We think that in questions this is lower.

Learning rules
Instead of writing handmade rules, it was proposed to use machine learning algorithms to detect them [34]. The idea is to create a training set where questions are tagged with entities (E), relations (R), classes (C), variables (V) and "none" tags like this:

none	V-B	C-B	none	none	E-B	E-I	R-B	.
By	which	countries	was	the	European	Union	founded	?

where E-B indicates that the entity is beginning and E-I that it is continuing (this way phrases with more words are labeled). Once a training corpus is constructed it is used to build a phrase tagger. The phrase tagger build in Xser[34] uses as features POS-tags, NER-tags and the words of the question itself. This way one can construct a tagger that is able to identify entities, relations, classes and variables in a given question and learn automatically rules to do that starting from the training data. Since POS-tags are used as one of the features, the handmade rules presented above should be automatically founded.
The disadvantage is that a training corpus must be created. The advantage is that no handmade rules have to be found.

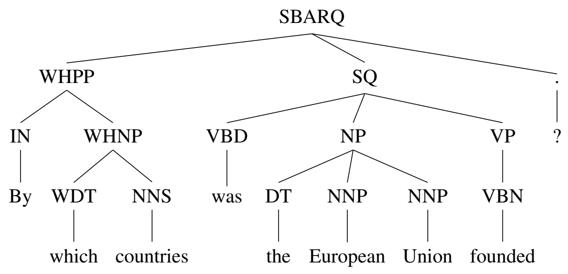
4.3. Parsers

A parser is based on a formal grammar. A formal grammar consists of symbols (words) and production rules that are used to combine the symbols. Given a sentence, a parser computes the combination of production rules that generate the sentence according to the underlying grammar. Examples will be given in the next section. Note that there are different types of formal grammars. The systems participating to the QALD challenges use parsers based on different types of grammars. We present them below and show their advantages.

[9]https://opennlp.apache.org

4.3.1. *Parsers based on phrase structure grammars*

Some parsers rely on phrase structure grammars. The idea of phrase structure grammars is to break down a sentence into it's constituent parts. As an example consider the parsing tree of the question "By which countries was the European Union founded?" returned by the Stanford Parser[10]:



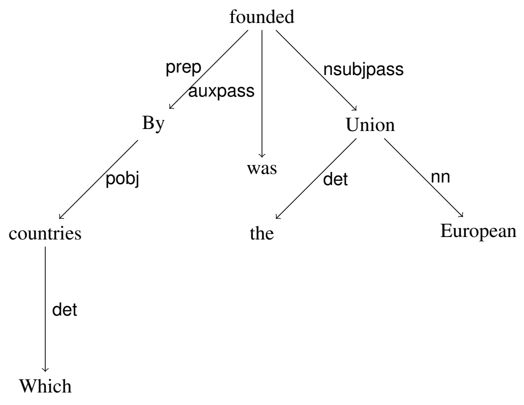
At the bottom of the tree are the words in the question and the corresponding POS tags. The tags above denote phrasal categories like: noun phrase (NP), verb phrase (VP), main clause of a wh-question (SQ) and direct question introduced by a wh-phrase (SBARQ). In phrase structure grammars the production rules define how the POS tags can be combined to form phrasal categories and how to combine phrasal categories to new one.
This types of trees are used similarly to POS tags, i.e. one tries to find some graph patterns that map to instances, properties or classes with high confidence. Differently from POS tags, one can deduce the dependencies between entities, relations and classes. Parsers of such type are used in the QA systems Intui2 [12], Intui3 [13] and Freya [9].

4.3.2. *Parsers based on dependency grammars*

The idea behind dependency grammars is that the words in a sentence depend from each other, i.e. a word "A" depends from a word "B". "B" is called the head (or governor) and "A" is called the dependent. Moreover, parsers also generally indicate the type of relation between "A" and "B".

Standford dependencies, Universal dependencies
The following example shows the result of the Stanford dependency parser for the questions "By which countries was the European Union founded?":

[10]http://nlp.stanford.edu/software/lex-parser.shtml



The tree indicates for example that "founded" is the head of "By", or that "By" is the dependent of "founded". The Stanford parser also returns the type of the dependency, in the example above "prep" indicates that "by" is a preposition of "founded". Note that the Stanford dependency parser supports two types of dependencies: the Stanford Dependency and the Universal Dependency. The dependency tree above uses the Stanford Dependency.
The advantages of this type of dependency trees can be shown by looking at the example above. In the original question, the words in the relational expression "founded by" are not subsequent which makes them difficult to extract. This is not the case in the dependency tree where "by" is connected to "founded". This is the main reason why dependency representations are used for relation extraction. Moreover, the parsing tree contains grammatical relations like nsubj (nominal subject), nsubjpass (nominal passive subject), dobj (direct object), pobj (object of a proposition) and others which can be used to identify the relations between different phrases in the question.
A method to extract relations is to search the dependency tree for the biggest connected subtree that can be mapped to a property. In the example above this would be the subtree consisting of the nodes "founded" and "by". The arguments associated to a relation are then extracted searching around the subtree. This is for example used by gAnswer[39]. Another strategy is to first identify named entities in the dependency tree and choose the shortest path between them as a relation. In the example above these are the entities "country" and "European Union". This is used in [24].

Phrase dependencies and DAGs
While the Stanford dependency parser considers dependencies between words, the system Xser [34] con-

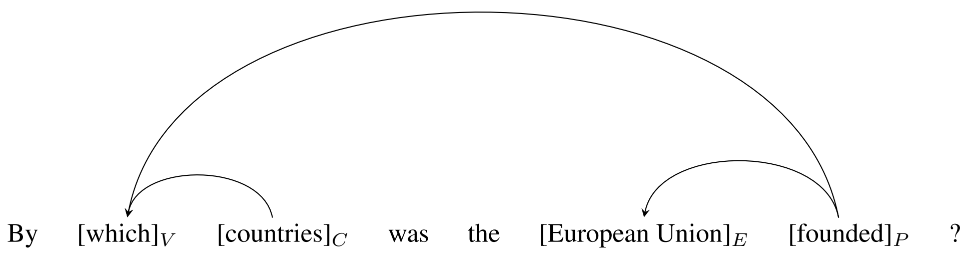


Fig. 2. Dependency relations using phrase dependencies

siders dependencies on a phrase level, namely between phrases referring to entities, relations, classes and variables as in 4.2. The following dependency relation is considered: relations are the head of the two corresponding arguments and classes are the head of the one corresponding argument (arguments are either entities, variables or other relations and classes). As an example, look at the dependency graph of the question "By which countries was the European Union founded?" in figure 2. The phrase "founded" is the head of the phrase "which" and the entity "European Union". The class "country" is the head of the variable "which". Note that this is not a tree, but a direct acyclic graph (DAG). To compute the DAG, Xser uses a SHIFT-REDUCED parser that is trained on a manually annotated dataset. The parser uses as features the POS tags of the words, the type of the phrase and the phrase itself. The advantage is that the parser learns automatically which is the relation between the phrases. The disadvantage is that one needs a manually annotated corpus.

4.4. Parsers that use a light semantic representation

TBSL [29] and BELA [31] use a parser based on lexical tree-adjoint grammars (LTAG). The idea of LTAG is that the symbols of the grammar do not correspond to words but to trees. The LTAG describes how to combine these trees to obtain syntactically valid statements. Note that to parse a sentence there must be a tree associated to each word in the sentence. Since questions to large ontologies contain too many possible words TBSL and BELA consider domain-independent and domain-dependent expressions. The first includes light verbs (to be, to have, give me), question words (what, which, how many) and other determiners (some, all, more/less than). For the domain-dependent expressions possible candidate trees are automatically generated based on the POS tag.
The main point in the approach used by TBSL and BELA is that each tree is paired with a semantic representation. This is used in the following way. First the question is parsed, i.e. a combination of trees is computed that generates the question. Following this combination, the semantic interpretations associated to each tree are combined into a semantic interpretation of the whole question. This is then translated to query templates, i.e. queries with slots (corresponding to the domain-dependent expressions) that have to be filled with resources of the ontology. This means that a whole query template is generated based only on a syntactic analysis and some light, ontology-independent semantic representations. The main advantage of this approach is that it is able to deal with superlatives and comparatives.

4.5. Summary

Table 3 shows which strategy is used by the different engines for question analysis. We put a question mark if it is not clear from the description of the publication if a technique is used.

5. Phrase mapping

This step starts with a phrase (one or more words) *s* and tries to find, in the underlying ontology, a set of resources which correspond to *s* with high probability. Note that *s* could correspond to an instance, a property, or a class. As an example, the phrase *EU* could correspond to the DBpedia resources *dbr:European_Union* but also to *dbr:University_of_Edinburgh* or *dbr:Execution_unit* (a part of a CPU).
We first sketch the general strategy used to solve this problem. RDF Schema introduces the property

Table 3

Techniques use in the question analysis task

	NER	NE n-gram startegy	other tools to detect NE	POS hand-made	POS learned	Parser structural grammar	Dependency parser	Phrase dependencies and DAG	Parser with semantic representation
BELA				x					x
CASIA		x			x		x		
DEANNA		x		x			x		
FREyA						x			
gAnswer							x		
Intui2						x			
Intui3	x			x					
ISOFT			x	x			x		
PowerAqua				x					
QAKiS	x	x							
QAnswer		x					x		
RTV	?						x		
SemGraphQA		x					x		
SemSeK		x				x	x		
SINA		x							
TBSL				x					x
Xser	?				x			x	

rdfs:label[11] to provide a human-readable version of a resource's name. Moreover, multilingual labels are supported using language tagging. For example, the following triples appear in DBpedia:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>;
PREFIX dbr: <http://dbpedia.org/resource/>;
dbr:European_Union rdfs:label "European␣Union"@en .
dbr:European_Union rdfs:label "Europäische␣Union"@de .
dbr:European_Union rdfs:label "Union␣européenne"@fr .
```

saying that the human-readable version of *dbr: European_Union* in french is *Union européenne*. By using the RDF Schema convention the resources corresponding to a phrase s can be found by searching all resources r in the ontology whose label $label(r)$ is equal to or contains s.

[11] @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

Several problems can arise in this step. They can be grouped in two categories. The first category contains problems that arise because the phrase s, as a string, is similar to $label(r)$ but not a strict subset. For example, s could be misspelled or the order of the words in s and $label(r)$ could be different.
The second category contains problems that arise because s and $label(r)$ are completely different as strings but similar from a semantic point of view. This can happen for example when the questions contains abbreviations like *EU* for *dbr:European_Union*, nicknames like *Mutti* for *dbr:Angela_Merkel* or relational phrases like *is married to* corresponding to the DBpedia property *dbo:spouse*. All these problems arise because the vocabulary used in the question is different from the vocabulary used in the ontology. The term "lexical gap" is used to refer to these problems. In the following, we want to provide an overview of the tech-

niques used in the QA systems to deal with these problems.

5.1. *Dealing with string similarity*

One possibility to deal with misspelling is to measure the distance between the phrase s and the labels of the different resources of the ontology. There are a lot of different distance metrics that can be used. The most common one are the Levenshtein distance and the Jaccard distance. The Levenshtein distance quantifies how dissimilar two strings are by counting which is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to transform one string into another. For example if the string s is *Europe Union* the Levensthein distance to *Europe* (6 edits) is bigger then to *European Union* (2 edits). The Jaccard distance is the ratio between the non-shared words of two strings and the total number of distinct words in the two strings. For example the distance between the string *European Union Council* and *Council of the European Union* is lower than the distance to *European Union Satellite Centre*.
A complementary way to retrieve the resources whose labels are similar to s is to use Lucene[12]. Generally, Lucene is used to index a list of documents. Given a phrase s Lucene returns a ranked list of documents that are relevant to the searched string. In the context of phrase mapping, instead of documents, the set of all labels of an ontology are stored using a Lucene index. Given a phrase s the most relevant labels are returned. The ranking is based on tf-idf weighting and a vector space model. Roughly speaking tf-idf gives weights to the different words in the labels. Words that are common between a lot of labels are down weighted and words which are specific for one label are up weighted. The vector space model then is used to combine the tf-idf weights in the case the phrase s contains multiple words. This means that the top ranked labels are then the labels for which the phrase s is very characteristic. Moreover, Lucene also provides fuzzy searches which return words similar to the given one based on the Levenshtein Distance or another edit distance. Engines that use Lucene are PowerAqua [20] and SemSeK [2]. The advantage of using this technique is that it is highly performant and one has ranked resources. The main disadvantage is that the index has to be kept synchronized with the corresponding ontology.

[12]http://lucene.apache.org/

5.2. *Dealing with semantic similarity*

In this section we want to consider the phrase mapping problem when s and $label(r)$ have only a semantic relation.

5.2.1. *Databases with lexicalizations*

One frequent approach is to use a lexical database. Examples for such databases are WordNet and Wiktionary. The strategy is to first expand s by synonyms $s_1, ..., s_n$ found in the lexical database. Then $s_1, ..., s_n$ are used to retrieve the resources instead of s. For example WordNet returns for *EU* the synonyms *European Union, European Community, EC, European Economic Community, EEC, Common Market, Europe* which are then used to retrieve the corresponding resources in the ontology. More precisely WordNet groups words in sets of roughly synonymous words called synsets.
For example, the word *EU* is contained in two synsets. The first was described above and refers to *"an international organization of European countries formed after World War II to reduce trade barriers and increase cooperation among its members"*. The second synset contains *europium, Eu, atomic number 63* and refers to *"a bivalent and trivalent metallic element of the rare earth group"*.
To find the synonyms of *EU* all synsets have to be merged together since it is not clear to which meaning one refers. WordNet also organizes the synsets in a taxonomy. For example, Y is a hypernym of X if every X is (a kind) of Y. On the other side Y is a hyponym of X if every Y is (a kind) of X. This can be used to further expand s not considering only the synsets of s but also its hypernyms and hyponyms. WordNet is used for example in TBSL [29], PowerAqua [20] and SemSek [2]. The main advantage is that more mappings can be resolved improving recall. The disadvantage is that the expansion of s increases the number of possible corresponding resources. This can affect precision and has the side effect that the disambiguation process becomes computationally more heavy. Moreover lexical databases are often domain-independent and cannot help for domain-specific mappings.

5.2.2. *Redirects*

Another way to collect new labels of a resource is to follow, if present, the owl:sameAs links. The labels in the connected ontology can be used then in the same way as in the original ontology. Moreover, also the anchor texts of links, mapping to an ontology resource, can also be used. The labels generated by the anchor texts of the Wikipedia links are for example contained in DBpedia lexicalizations[13].

5.2.3. A database with relational lexcalizations (PATTY)

PATTY [24] is a Database with relational lexicalizations. The main objective of this system is to collect natural language expressions and group them if they refer to the same relation. These groups are called "pattern synsets" in analogy of Wordnet. Moreover, the pattern synsets are organized in a taxonomy like synsets in WordNet. An example of such a pattern synset is *(is album, [[num]] album by, was album, [[det]] album with, [[adj]] album by)* where *[[num]]* corresponds to a cardinal number, *[[det]]* corresponds to a determiner and *[[adj]]* corresponds to an adjective (for this reason they are called "patterns"). Examples where different phrases express the same relation are: "Beatles*'s album* Revolver ...." , "Revolver is a *1966 album by* the Beatles" , "Revolver is *an album with* the Beatles", and "Revolver is the *seventh album by* the Beatles".
To extract the natural language expressions corresponding to relations, a text corpus is scanned searching named entities contained in a fixed ontology O. Assume that a sentence with two named entities is found. Then the dependency graph of the sentence is computed and a phrase is extracted from it connecting the two named entities. This leads to a list of patterns and pairs of entities connected by the pattern. The main idea is to put two patterns in the same "pattern synset" if the pairs of entities connected by them is equal. This means that all patterns in the same "pattern synset" can be used to relate the same pairs of named entities. Moreover, the constructed "pattern synset" are arranged in a taxonomy in the following way. If A and B are two "pattern synsets" then the patterns in A subsume the patterns of B if all named entities linked by the relations in B are also linked by relations in A. Note that in this approach the properties of the ontology O are not used. The ontology is only used to identify the named entities. In particular a "pattern synset" does not need to correspond to any property in the ontology.
PATTY can be used as standard database with lexicalizations as in 5.2.1 and shares their advantages and disadvantages. PATTY is for example used by Xser [34].

[13]http://wiki.dbpedia.org/lexicalizations

5.2.4. Finding mapping using extracted knowledge

There are different tools that extract binary relations expressed in natural language from text corpora. An example of such a binary relation is *("Angela Merkel", "is married to", "Joachim Sauer")*. Note that it is a relation expressed in natural language so the subject, object and predicate are not mapped to any ontology. Tools that can extract this type of triples are for example ReVerb [14], TEXTRUNNER[37], WOE[33] and PATTY [24]. There are two approaches that use this data to find natural language representations of properties in an ontology. Both assume that the subject and object of the binary relation are mapped to instances in an underlying ontology.
The first was described in [5]. First the relational phrase is normalized to an expression rel. In a second step the classes of the subject and object are added obtaining $rel[class1, class2]$, i.e. in the example above *married[Person, Person]*. Then all entity pairs are computed that are connected in the text by the relation rel and that matches the classes. Denote this set as $Sup(rel[class1, class2])$. Moreover, for all properties in the ontology the set of connected entity pairs are computed. For a property p we denote this set as $Sup(p)$. The relation $rel[class1, class2]$ is then aligned to the property p if the domain and range of both agree and $Sup(rel[class1, class2]) \cap Sup(p) \neq \emptyset$. This technique was used by CASIA [18]. The main disadvantage here is that the data can be very noisy.
The second was described by [39]. It starts as the first approach with a fix natural language relation rel that was extracted from a text and a set of entity pairs that are connected by rel and that occur in the underling ontology. Assume that the set of entity pairs is $\{(v_1, w_1), ..., (v_n, w_n)\}$. Then for each pair (v_i, w_i) the shortest path $Path(v_i, w_i)$ connecting v_i to w_i is searched in the ontology. This way one has a collection of paths $PS = \cup_{i=1}^{n} Path(v_i, w_i)$. The idea is that if there is a path P that is very frequent then it is a good candidate for expressing rel. If P has length one, then we have found a candidate property for the relation rel. If P has length greater than one the relation rel probably cannot be expressed by a single property in the ontology, but by the composition of the properties in P. An example of such a relation is *uncle of* which does not correspond directly to a DBpedia property, but that can be expressed using a concatenation of the property *dbpedia-owl:child*. This technique was used by gAnswer [39].

5.2.5. Finding mappings using large texts

There are two methods that use large texts to find natural language representation for phrases. The first is the core of the BOA framework [16]. The BOA framework takes as an input a property p contained in an ontology and tries to extract natural language expressions for p from a large text corpus. To do that BOA extracts from the ontology the subject-object pairs *(x,y)* that are connected by the property p. Then the text corpus is scanned and all sentences are retrieved that contain the pairs *(label(x), label(y))*. At the end it extracts the segments of text between $label(x)$ and $label(y)$, or $label(y)$ and $label(x)$. The idea is that these text segments are a natural language representation of the property p. These text segments are stored together with the range and domain of p and ranked such that top ranked patterns more likely are a natural language representation of p. Using this data, a relational phrase r can be mapped to a property p by searching a similar text fragment in the BOA framework. This technique is for example used in TBSL [29]. Moreover Qakis[6] uses the same approach in a more restricted fashion and without using BOA. Also QAnswer [27] uses this approach, but the relational phrase is extracted using dependency trees. The big advantage of this approach is that the extracted relational phrases are directly mapped to the corresponding properties. The relational expressions found are tightly connected to the ontology, differently to PATTY where the relations are more ontology-independent.
Another approach based on large texts uses the tool word2vec[14]. The main idea of word2vec is to associate a vector to each word using a machine learning technique. The nice thing about these vectors is that they encode part of the semantic of the words. We give two examples that appear in literature. Experimental results showed that the closest vectors to the vector of France *vec(France)* are the vectors *vec(Spain), vec(Belgium), vec(Netherlands), vec(Italy)*. Moreover, the vector *vec(queen)* is very near to the vector *vec(king)-vec(man)+vec(woman)*. Word2vec was used by CASIA [18] to map phrases to classes in the following way. First a phrase, that was expected to relate to a class, was converted using word2vec into a vector v. The classes whose vectors are more similar to v are then considered as possible candidates. An analogous tool is Explicit Semantic Analysis (ESA)[15]. This was used by BELA [31]. In general, one speaks about word embeddings, i.e. words are embedded in a vector space as vectors.

[14]https://code.google.com/p/word2vec/
[15]http://code.google.com/p/dkpro-similarity-asl/

5.3. Others

Some engines use some other tools for the phrase mapping task namely: DBpedia lookup[16] and the Wikipedia Miner Tool[17]. Both these tools can only be used for the DBpedia ontology. DBpedia lookup, for examples uses the number of inlinks for ranking, i.e. the more Wikipedia pages link to the Wikipedia page of one resource the more important it is. But at the core, DBpedia lookup is a lucene index over the labels of the resources enriched with the anchor texts of the redirect links of Wikipedia. It is for example used by gAnswer [39].
The Wikipedia Miner Tool is a tool which is completely related to Wikipedia and cannot be reused for any other ontology than DBpedia. It is for example used by Xser [34].

5.4. Summary

Table 4 gives an overview showing which technique is used by which engine for phrase mapping.

The important point in this step is that one has to find a balance between selecting as few candidate resources as possible to improve precision and time performance, and select enough candidates such that also the relevant one is selected to improve recall.

6. Disambiguation

Two problems can arise from the previous steps. The first is that from the question analysis step the segmentation and the dependencies between the segments are ambiguous. The second is that the phrase mapping step returns multiple possible resources for one phrase. This adds another ambiguity. This section explains how question answering systems deal with these ambiguities.
First the general strategy is sketched to solve these problems. Due to the ambiguities for one question, a QA system generates many possible interpretations.

[16]https://github.com/dbpedia/lookup
[17]http://wikipedia-miner.cms.waikato.ac.nz

Table 4

Techniques used in the phrase mapping task

	String similarity	Lucene index or similar	WordNet/Wiktionary	Redirects	PATTY	Using extracted knowledge	BOA or similar	Word embeddings	other
BELA	x	x	x	x				x	
CASIA				x		x		x	
DEANNA									
FREyA	x		x						
gAnswer[18]				x		x			x
Intui2									
Intui3			x	x					x
ISOFT		x			x			x	
PowerAqua		x	x	x					
QAKiS							x		
QAnswer	x	x	x	x			x		
RTV		x						x	
SemGraphQA		x	x	x					
SemSeK		x	x	x				x	
SINA									
TBSL	x	x	x				x		
Xser					x				x

All systems do a "local" form of disambiguation to choose between the possible interpretations. Mainly two features are used for this. The first is the (string or semantic) similarity of the label of the resource and the corresponding phrase. The second is a type consistency check between the properties and their arguments. The first feature is used to rank the possible interpretations, the second to exclude some. These features are "local" in a sense that only the consistency between two resources that are directly related is checked. Assume for example a user inputs the question "Which instruments do the members of Europe play?". In this case "Europe" can be the continent but also the band called "Europe". In this case "Europe" is related to "members" which is related to "instruments". Using only local features the relation between "instruments" and "Europe" will be ignored leading probably to a wrong interpretation of the question.
The advantage is that "local" disambiguation is very easy and fast. Moreover, it is often very powerful. A main disadvantage is that actual ontologies often do not contain domain and range information of a property so that the type consistency cannot be done. Sometimes engines directly check if two or three resources are in the same triple, but this is time consuming.
Now we present techniques that also use "global" information to address the disambiguation. Since these techniques are very different from each other we do not provide a classification.

6.1. Graph search

The QA system gAnswer [39] introduces the following strategy to address the ambiguity that arises in the phrase mapping phase. It assumes that in the question analysis step the intention of a question q can be translated into a graph G. This contains a node for each variable, entity and class in the question, an edge for each relation. Moreover, it is semantically equivalent to q.
The ambiguity that arises in the phrase mapping step carries over to an ambiguity of vertices and edges of G. The idea of gAnswer is to disambiguate the vertices and edges of G by searching into the ontology a subgraph isomorphic to G. This is done such that the corresponding vertices and resources correspond to the

segments of q with high probability. This is achieved by assigning a score to each possible match, which is proportional to the distance between the labels of the resources and the segments of the question. The top-k matches are retrieved.
The main disadvantage of this approach is that one needs to find the dependencies of the resources in the question analysis step. The main advantage is that although checking sub-isomorphism is NP-hard, during experiments it performs faster than most of the other approaches.

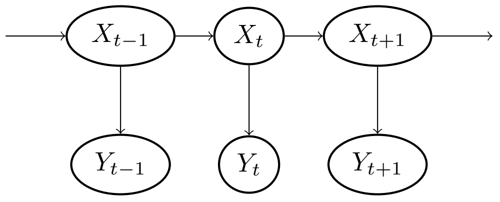
6.2. Hidden Markov Model (HMM)

Hidden Markov Models (HMM) are used by SINA [28] and RTV [17] to address the ambiguity that arises in the phrase mapping phase. The idea behind this strategy is sketched using an example. Assume the question is: "By which countries was the EU founded?". In a Hidden Markov Model (HMM) one has two stochastic processes $(X_t)_{t\in\mathbb{N}}$ and $(Y_t)_{t\in\mathbb{N}}$ where only the last one is observed. The possible values of the random variables X_t are called hidden states whereas the possible values of the random variables Y_t are called observed states. For the example above, the set of observed states is *{"countries", "EU", "founded"}*, i.e. the set of segments of the question that have an associated resource. The set of hidden states is *{dbo:Country, dbr:Euro, dbr:European_Union, dbr:Europium, dbp: founded, dbp:establishedEvent}*, i.e. the set of possible resources associated to the segments.
In a Hidden Markov Model the following dependency between the random variables are assumed:

- $P(X_t = x_t|X_0 = x_0; ...; X_{t-1} = x_{t-1}; Y_0 = y_0; ...; Y_t = y_t) = P(X_t = x_t|X_{t-1} = x_{t-1})$, i.e. the value of a variable X_t only depends on the value of X_{t-1} which means that $(X_t)_{t\in\mathbb{N}}$ is a Markov Chain;
- $P(Y_t = y_t|X_0 = x_0; ...; X_t = x_t; Y_0 = y_0; ...; Y_{t-1} = y_{t-1}) = P(Y_t = y_t|X_t = x_t)$, i.e. that the value of Y_t depends only from X_t.

If one indicates the conditional dependencies with an arrow one gets the diagram in figure 3.

In the context of disambiguation this means that the appearance of a resource at time t depends only from the appearance of another resource at time $t-1$ and that the segments appear with some probability given a resource. The disambiguation process is reduced to the case of finding the most likely sequence of hidden



$$\begin{tikzcd}
{} \arrow[r] & X_{t-1} \arrow[r] \arrow[d] & X_t \arrow[r] \arrow[d] & X_{t+1} \arrow[r] \arrow[d] & {} \\
& Y_{t-1} & Y_t & Y_{t+1} &
\end{tikzcd}$$

Fig. 3. Conditional dependencies in a Hidden Markow Model

states (the resources) given the sequence of observed states (the segments). This is a standard problem that can be solved by the Viterbi Algorithm.
To complete the modeling one needs to indicate three more parameters:

- the initial probability, i.e. $P(X_0 = x)$ for $x \in X$;
- the transition probability, i.e. $P(X_t = x_1|X_{t-1} = x_2)$ for $x_1, x_2 \in X$;
- the emission probability, i.e. $P(Y_t = y|X_t = x)$ for $x \in X$ and $y \in Y$.

These can be bootstrapped differently. In SINA the emission probability is set according to a string similarity measure between the label of the resource and the segment. In RTV the emission probabilities are estimated using word embeddings. In SINA the initial probabilities and the transition probabilities are estimated based on the distance of the resources in the ontology and their popularity. Retrieving the distance between all resources is computationally very expensive making this approach slow. In RTV the initial and transition probabilities are set to be uniform for all resources making the estimation fast but more inaccurate.
An advantage of this technique is that it is not necessary to know the dependency between the different resources but only a set of possible resources.

6.3. Integer Linear Program (ILP)

In the question answering system DEANNA [35] it was proposed to set up an Integer Linear Program (ILP) to solve the disambiguation task. This technique addresses the ambiguity of the phrase mapping phase and some ambiguity that arises during the segmentation.
The ILP uses boolean variables to indicate for example if a segment of the question is chosen, if a resource corresponding to a segment is chosen or whether a segment corresponds to a property or an instance. The constraints include conditions such that the chosen segments do not overlap, such that if a segment is cho-

sen then one corresponding resource must be chosen and so on. The optimization function includes three terms. The first increases if the label of a resource is similar to the corresponding segment. The second increases if two selected resources often occur in the same context. The third tries to maximize the number of selected segments. Note that the second term makes the disambiguation process work.
The main disadvantage is that some dependencies between the segments have to be computed in the question analysis phase.

6.4. *Markov Logic Network*

The question answering system CASIA [18] uses a Markov Logic Network (MLN) for the disambiguation task. The idea is to define some constraints using first-order logic formulas. MLN allows to consider some of them as hard constraints that must be fulfilled and others as soft constraints, i.e. if they are not satisfied a penalty is applied. In this case both the ambiguities that arise in the question analysis and phrase mapping stage are resolved.
Examples of hard constraints are: if a phrase of the question is chosen then one of the corresponding resources must be chosen, or that the chosen phrases cannot overlap. Examples for soft constraints are: if a phrase has a particular POS tag then it is mapped to a relation, the resource whose label is most similar to the corresponding phrase must be chosen.
The hard constraints of a MLN have a similar behavior as the constraints in a ILP while the soft constraints allow more flexibility. The penalty for the soft constraints are learned in a training phase.
The advantage of a MLN is that they allow more flexibility than an ILP in choosing the constraints. However, a training phase is needed.

6.5. *Structured perceptron*

The engine Xser [34] uses a structured perceptron to solve the disambiguation task. The idea is to consider features during disambiguation such as: the similarity of a phrase and the corresponding resource, the popularity of a label for a resource, the compatibility of the range and domain of a property with the types of the arguments, the number of phrases in the question that are in the same domain. In a training phase for each of the features f a weight w is computed such that the expected configuration z fulfills:

$$z = argmax_{y \in Y(x)} w \cdot f(x, y)$$

where x is the question and $Y(x)$ is the set of possible configurations of the resources and dependency relations. For a new question the configuration is chosen that maximizes the expression above. In this approach the ambiguity that arises in the phrase mapping phase is resolved. However, a training phase is needed.

6.6. *User feedback*

There exist situations in which the question answering engine cannot do the disambiguation automatically. This can happen because the disambiguation technique used by the engine does not suffice or because the question is really ambiguous. Therefore, some systems ask the user to resolve the ambiguity by choosing between some proposed resources. A system that relies heavily on user feedback for the disambiguation is Freya [10].

6.7. *Summary*

Table 5 gives an overview of the different techniques used by the QA systems for disambiguation.

7. Query construction

In this section, we describe how each QA system combines the techniques presented in the previous sections to construct a SPARQL query. A problem arises during the query construction, the so called "semantic gap". Assume for example that a user asks the question: "Which countries are in the European Union?". One would probably assume that in the ontology there are triples like:

```
dbr:Greece dbp:member dbr:European_Union .
dbr:France dbp:member dbr:European_Union .
```

But this is not the case, in DBpedia the requested information is encoded as:

```
dbr:Greece dct:subject
          dbc:Member_states_of_the_European_Union .
dbr:France dct:subject
          dbc:Member_states_of_the_European_Union .
```

Table 5

Techniques use in the disambiguation task

	Graph search	HMM	LIP	MLN	Structured perceptron	User feedback
BELA						
CASIA				x		
DEANNA			x			
FREyA						x
gAnswer	x					
Intui2						
Intui3						
ISOFT						
PowerAqua						
QAKiS						
QAnswer						
RTV		x				
SemGraphQA						
SemSeK						
SINA		x				
TBSL						
Xser					x	

So instead of a property "dbp:member" DBpedia uses the class "dbc:Member_states_of_the_European _Union" to encode the information. The "semantic gap" refers to the problem that the ontology encodes an information differently from what one could deduce from the question. Another example is the question: "By which countries was the European Union founded?". Here one probably expects that the information is encoded using a triple like:

```
dbr:West_Germany dbp:founded dbr:European_Union
```

But the requested information is encoded as:

```
dbr:European_Union dbp:establishedEvent
                                   dbr:Treaty_of_Rome .
dbr:Treaty_of_Rome dbp:parties dbr:West_Germany,
     dbr:Belgium, dbr:France, dbr:Italy,
     dbr:Luxembourg, dbr:Kingdom_of_the_Netherlands
```

These cases show that in general it is impossible to deduce the form of the SPARQL query knowing only the question. Therefore, we classify the approaches for the query construction based on how the SPARQL query form is deduced. We distinguish between approaches where the SPARQL query form is based on templates, approaches where it is deduced from the question analysis phase, where it is deduced using machine learning techniques or where it is deduced using only semantic information. The last subsection describes the approach of SemSek that does not generate a SPARQL query.

7.1. Query construction using templates

Some engines use templates to generate the SPARQL query, i.e. a set of predefined queries with some slots that have to be filled. The system QAKiS [6] restricts to select queries with only one triple. The system ISOFT [25] uses a small set of templates to generate SPARQL queries: these include ASK queries over one triple, some simple SELECT queries with one or two triples and templates that use a COUNT, ORDER BY or FILTER expression containing only one triple. Also PowerAqua [20] assumes that the input question can be reduced to SELECT queries with one or two triples. In all three cases the phrases to fill the template are identified in the question analysis step. These are converted to RDF triples in the phrase mapping phase. After some local disambiguation the SPARQL query is constructed. The disadvantage is of course that not all questions can be treated using templates.

7.2. Query construction using information from the question analysis

Most of the systems start with the information obtained in the question analysis part and deduce from it the form of the SPARQL query.
Freya and Intui3 [13] start from the segmentation of the question. In the phrase mapping phase some segments have an associated resource. These resources are then combined into triples respecting the order of the segments in the question. If necessary some additional variables between the identified segments are added, for example if one relation is followed by another relation.
DEANNA [36] selects some triples phrase candidates in the question analysis phase using regular expressions over POS tags. These are mapped to resources in the phrase mapping phase. In the disambiguation phase the best phrase candidates and the best phrase mappings are chosen using a LIP. This returns a set of triples that is then used to construct a SELECT query.
Intui2 [12] starts from the parse tree. Using some predefined patterns on the parse tree, some phrases are extracted and labeled as subject/object or predicate. These are combined into triples following the parse tree and doing local disambiguation at each step. The triples are then combined into a SELECT query.
The QA systems gAnswer, QAnswer, RTV and SemGraphQA start with a dependency tree. In gAnswer [39] first the relations and the associated arguments are deduced. A graph G is constructed which has an edge for each relation and a vertex for each argument. The graph G reflects the structure of the final SPARQL query. The relations and arguments are mapped to possible resources. The right resources are identified using the sub-isomorphism strategy described in section 6.1. Then the SPARQL query is constructed.
QAnswer [27] first scans the dependency tree to find subgraphs of tokens that correspond to some resources. This way many graphs with associate resources are created. Then a local disambiguation is performed. The top ranked graph is chosen and the SPARQL query is constructed from this graph.
RTV [17] uses the dependency graph to construct an ordered list of alternating properties and non properties. The corresponding resources are searched and disambiguated using a HMM. From this sequence the SPARQL query is generated.
SemGraphQA [4] simplifies the dependency tree by keeping only relevant dependency relations (omitting for example determiners and auxiliaries). This way a graph G is generated. SemGraphQA generates new graphs by associating to each node of G the label "relation" and "entity" (corresponding to a class, a variable or an instance). If two "relation" nodes are connected a new "entity" node is inserted between them corresponding to a variable. Graphs are excluded when they are not connected, when there is an "entity" node that is not connected to a "relation" node and when a "relation" is not connected to two "entity" nodes. Then the possible resources are associated to each node. Finally, a local disambiguation is performed that selects one graph that is converted into a SPARQL query.
TBSL [29] and Bela [31] generate SPARQL queries with some slots during the question analysis phase. The slots are replaced by resource URIs in the phrase mapping phase. This way a set of SPARQL queries are generated. The best SPARQL query is chosen using local disambiguation t.
Xser [34] uses three different machine learning algorithms. The first and the second are claimed to be ontology independent. The first is used to determine the segments of the question corresponding to variables, properties, instances, and classes. The second is used to find the dependencies between the phrases as described in section 4.3.2. The third is used in the disambiguation phase, which is described in section 6.5 and is ontology dependent. Since the first two algorithms are claimed to be ontology independent this approach also constructs the form of the SPARQL by analyzing the question.
All these approaches share the same disadvantage. All of them make the implicit assumption that it is possible to deduce the structure of the SPARQL query from the structure of the question.

7.3. Query construction using machine learning

CASIA [18] uses a machine learning approach for the whole QA process. The question analysis phase is used to segment the question and to extract features like the position of a phrase, the POS tag of a phrase, the type of dependency in the dependency tree and some other. In the phrase mapping phase resources are associated to the segments and new features are extracted: the type of the resources and a score for the similarity between the segment and the resource. In the disambiguation phase the extracted features are used

in a MLN (as described in 6.4) to find the most probable relation between the segments and to find the most probable mapping. The detected relations are then used to generate the SPARQL query. The disambiguation phase must be retrained for each new ontology.
This approach allows more flexibility then the approaches described above. This can be used to address the "semantic gap". However, this comes with the price of a training phase.

7.4. Query construction relaying on semantic information

The question answering system SINA [28] was developed to deal primary with keyword queries, i.e. instead of inserting the question "What is the capital of Belgium?" the user can also insert the keywords "capital Belgium". This implies that the relation between the different resources is not explicit like in a natural language question. In this case the dependencies between the resources have to be derived using the ontology.
SINA first segments the question and finds associated resources. These are disambiguated using a HMM. Once the resources are determined, SINA constructs the query in the following way. For each instance or class, a vertex is created. For each property, an edge is created. The edges are used to connect the vertices if the types of the range and domain allow it. If not, then either one or two new vertices are created that correspond to variables. Note that the combinatorics could allow more than one graph. In this case they are all considered because it is not clear which one reflects the user's intention. Moreover, at the end of the process it is possible that one gets unconnected subgraphs. In this case for each pair of vertices in two fixed subgraphs the set of possible properties that can connect them is computed and taken into account. All the possible graphs are translated to a SPARQL query and executed.
The advantage of this strategy is that the graph is constructed starting from the underlying ontology and not using the syntax of the question. The disadvantages are that this process is computationally complex and that the syntax of the question is not respected. For example, the system will not see any difference between the questions "Who is the mother of Angela Merkel?" and "Angela Merkel is the mother of who?".

7.5. Approach not using SPARQL

SemSek [2] does not generate a SPARQL query but instead navigates through the ontology by dereferencing resources. Consider the following example: "In which city was the leader of the European Union born?". SemSek first identifies a "central term" in the question, in this case "European Union". Using the dependency tree and starting from the "central term" an ordered list of potential terms corresponding to resources is generated. In the example above the list would be: "European Union", "leader", "born", "city". Then candidate resources for the first term (for example dbr:European_Union) are searched and the corresponding URI http://dbpedia.org/resource /European_Union is dereferenced to search all corresponding properties. These are compared with the second term in the list. The object is considered if one of the property is similar (as a string or semantically) to the second term in the list (like dbp:leaderName or dbp:leaderTitle). This generates two new exploring directions in the algorithm. Otherwise the actual direction is not further analyzed. SemSek then continues like this and searches the right answer in the ontology graph.

8. Conclusions

This analysis of QA systems shows two things. First, most of the systems have many common techniques. For example, a lot of systems use similar techniques in the question analysis and phrase mapping task. Second, QA systems generally concentrate on improving only some techniques and leaving aside others. For example, a lot of systems use only local disambiguation techniques, others use only basic techniques for the phrase mapping task.
We conclude that there is a need of an architecture for the integration of different approaches. This should offer plug-ins to solve the different problems that arise in the QA process. This will also allow the community to contribute to new plug-ins in order to either replace existing approaches or create new ones. The aim is that a QA system should not always be build from scratch such that research can focus more on single tasks. As a side effect it will become easier to compare techniques used to solve the same task.
We want to present soon such an architecture together with the first plug-ins. These plug-ins should satisfy the constraints of the Semantic Web. They should be

ontology independent, scalable, possible multilingual and require as few efforts as possible to set up a QA system over a new dataset.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642795.

References

[1] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/5/documents/qald-5_results.pdf`.

[2] Nitish Aggarwal and Paul Buitelaar. A system description of natural language query over dbpedia. *Proc. of Interacting with Linked Data (ILD 2012)[37]*, pages 96–99, 2012.

[3] Petr Baudiš and Jan Šedivỳ. QALD challenge and the yodaQA system: Prototype notes.

[4] Romain Beaumont, Brigitte Grau, and Anne-Laure Ligozat. Semgraphqa@ qald-5: Limsi participation at qald-5@ clef. CLEF, 2015.

[5] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[6] Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. Qakis: an open domain qa system based on relational patterns. In *11th International Semantic Web Conference ISWC 2012*, page 9. Citeseer, 2012.

[7] Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL (1)*, pages 423–433. Citeseer, 2013.

[8] Philipp Cimiano, Vanessa Lopez, Christina Unger, Elena Cabrio, Axel-Cyrille Ngonga Ngomo, and Sebastian Walter. Multilingual question answering over linked data (qald-3): Lab overview. In *Information Access Evaluation. Multilinguality, Multimodality, and Visualization*, pages 321–332. Springer, 2013.

[9] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. In *LREC*, 2010.

[10] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Freya: An interactive way of querying linked data using natural language. In *The Semantic Web: ESWC 2011 Workshops*, pages 125–138. Springer, 2012.

[11] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 121–124. ACM, 2013.

[12] Corina Dima. Intui2: A prototype system for question answering over linked data. *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF*, 2013.

[13] Corina Dima. Answering natural language questions with intui3. In *Conference and Labs of the Evaluation Forum (CLEF)*, 2014.

[14] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[15] Sébastien Ferré. squall2sparql: a translator from controlled english to full sparql 1.1. In *Work. Multilingual Question Answering over Linked Data (QALD-3)*, 2013.

[16] Daniel Gerber and A-C Ngonga Ngomo. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction@ ISWC*, volume 2011, 2011.

[17] Cristina Giannone, Valentina Bellomaria, and Roberto Basili. A hmm-based approach to question answering against linked data. *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF*, 2013.

[18] Shizhu He, Yuanzhe Zhang, Kang Liu, and Jun Zhao. Casia@ v2: A mln-based question answering system over linked data. *Proc. of QALD-4*, 2014.

[19] Guyonvarc'H Joris and Sébastien Ferré. Scalewelis: a scalable query-based faceted search system on top of sparql endpoints. In *Work. Multilingual Question Answering over Linked Data (QALD-3)*, 2013.

[20] Vanessa Lopez, Enrico Motta, Marta Sabou, and Miriam Fernandez. Poweraqua: A multi-ontology based question answering system–v1. *OpenKnowledge Deliverable D8. 4 Pp1*, 14, 2007.

[21] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating question answering over linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 21:3–13, 2013.

[22] Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. Aqualog: An ontology-driven question answering system for organizational semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):72–105, 2007.

[23] Vanessa Lopez, Victoria Uren, Marta Sabou, and Enrico Motta. Is question answering fit for the semantic web? a survey. *Semantic Web*, 2(2):125–155, 2011.

[24] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. Patty: a taxonomy of relational patterns with semantic types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1135–1145. Association for Computational Linguistics, 2012.

[25] Seonyeong Park, Hyosup Shim, and Gary Geunbae Lee. Isoft at qald-4: Semantic similarity-based question answering system over linked data. In *CLEF*, 2014.

[26] Camille Pradel, Guillaume Peyet, Ollivier Haemmerlé, and Nathalie Hernandez. Swip at qald-3: results, criticisms and lesson learned. *Valencia, Spain*, 2013.

[27] Stefan Ruseti, Alexandru Mirea, Traian Rebedea, and Stefan Trausan-Matu. Qanswer-enhanced entity matching for question answering over linked data. CLEF, 2015.

[28] Saeedeh Shekarpour, Edgard Marx, Axel-Cyrille Ngonga Ngomo, and Sören Auer. Sina: Semantic interpretation of user queries for question answering on interlinked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 30:39–51, 2015.

[29] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. *Proceedings of the 21st international conference on World Wide Web*, (639–648), 2012.

[30] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. Question answering over linked data (qald-4). In *Working Notes for CLEF 2014 Conference*, 2014.

[31] Sebastian Walter, Christina Unger, Philipp Cimiano, and Daniel Bär. Evaluation of a layered approach to question answering over linked data. In *The Semantic Web–ISWC 2012*. Springer, 2012.

[32] Matthias Wendt, Martin Gerlach, and Holger Düwiger. Linguistic modeling of linked open data for question answering. In *The Semantic Web: ESWC 2012 Satellite Events*, pages 102–116. Springer, 2012.

[33] Fei Wu and Daniel S Weld. Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127. Association for Computational Linguistics, 2010.

[34] Kun Xu, Yansong Feng, and Dongyan Zhao. Xser@ qald-4: Answering natural language questions via phrasal semantic parsing, 2014.

[35] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390. Association for Computational Linguistics, 2012.

[36] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. Robust question answering over the web of linked data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1107–1116. ACM, 2013.

[37] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.

[38] Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. Aida: An online tool for accurate disambiguation of named entities in text and tables. *Proceedings of the VLDB Endowment*, 4(12):1450–1453, 2011.

[39] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffer Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over rdf: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM, 2014.