QA over Linked Data using Stanford Dependencies

Part of the Question Answering Systems Project: Semantic Technologies in IBM
A project by Johannes Simon

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Language Technology Lab

Abstract

PAL is a Question Answering (QA) system for Linked Data. It is based on Hakimov's previous work as described in the paper *Semantic Question Answering System over Linked Data using Relational Patterns* [4]. It was developed during my participation in the *Question Answering Systems Project* [1] at the Language Technology Lab at TU Darmstadt [2]. This report describes how processing of natural-language questions was achieved using Stanford dependencies. It also introduces a web frontend that allows for interaction with the system. Evaluation is performed using the QALD-2 challenge [3].

[1] `https://www.lt.tu-darmstadt.de/de/teaching/lectures-and-classes/summer-term-2014/semantic-technologies-in-ibm-watson/`

[2] `https://www.lt.tu-darmstadt.de/`

[3] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=challenge&q=2`

Contents

1 Intruduction

Question Answering is an emerging research field that contrasts Information Retrieval in the traditional sense. Instead of documents that are retrieved, a QA system responds with direct answers to a question. This saves time for end users, e.g. doctors looking only for treatment indicator's in a patient's record. Reversely, it allows for searches in a larger amount of information within the same amount of time.

If we consider open-domain QA, most relevant data is only available in unstructured, natural-language (NL) text. However, some data is already structured, e.g. Linked Data (LD) from the DBpedia project [1]. As of 2014, the English DBpedia contains over 400 million facts about 3.7 million "things". [1]. Utilizing this structured information has proven to be useful in addition to processing NL text [2]. One advantages of linked data is its high precision, though this often comes at the expense of low recall. Take, for instance, the DBpedia project. For all facts, we know with high certainty that they are true. However, only a small portion of the information on Wikipedia is extracted into the DBpedia database. Therefore, Linked Data can be used to complement unstructured data for the task of QA.

SPARQL as interface to Linked Data

One possible way to query structured linked databases is to formulate the query using SPARQL [8]. It was standardized and later became an official recommendation by the World Wide Web Consortium (W3C) [1], and is therefore a good foundation to build on. Like other query languages for LD, and like the underlying data itself, SPARQL queries are mostly made up of triples of the form (subject, predicate, object). To translate NL into a SPARQL query, we therefore first need to translate the question into a set of triples. Every triple constraints the resulting answer set. For example, the question *Who is the author of "Deception Point"?* imposes the following SPARQL query:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

SELECT ?author WHERE {
   ?author dbpedia-owl:author dbpedia:Deception_Point .
}
```

Listing 1.1: SPARQL query for the question *Who is the author of "Deception Point"?*

[1] http://www.w3.org/blog/SW/2013/03/21/eleven-sparql-1-1-specifications-are-w3c-recommendations/

2 Approach and System Architecture

The system consists of two main parts. The first part maps a NL question to a SPARQL-like pseudo query. This pseudo query is not yet mapped to any ontology and contains only lexical information from the NL question. The second part of the system then maps this pseudo query to a specific ontology (provided by the SPARQL endpoint) and thus to a valid SPARQL query. Also provided is a web frontend which allows for easy interaction with the system.

2.1 System Requirements

To be able to install the system described in this report, you will require the following on your local computer:

- A Java Development Kit installation, at least version 1.6
- Git [1]
- Maven [2]
- A web server capable of deploying WAR-files, e.g. Tomcat [3] or Jetty [4]
- A WordNet 3.1 database [5]

2.2 Installation Instructions

See the following excerpt from my command line for installation instructions:

```
$ mkdir pal-installation && cd pal-installation
$ git clone https://github.com/johannessimon/pal.git
$ git clone https://github.com/johannessimon/pal-server.git
$ cd pal && mvn install -DskipTests
$ cd ../pal-server && mvn package -DskipTests
$ export WNHOME=/path/to/wordnet/3.1/
$ cp target/pal-server.war /path/to/webapps
```

Listing 2.1: Pseudo query for the question *Who is the author of "Deception Point"?*

Then start your web server if you haven't already done so, or restart if it does not support hot-deployment of the war file.

1 http://git-scm.com/
2 http://maven.apache.org/
3 http://tomcat.apache.org/
4 http://www.eclipse.org/jetty/
5 http://wordnetcode.princeton.edu/wn3.1.dict.tar.gz

2.3 Generating pseudo queries from natural language

A pseudo query is a set of triples, a set of type constraints and an identified question focus. For example, the question *Who is the author of "Deception Point"?* implies the triple ([Deception Point] [author] [?x]), the type constraint (?x a "author") and the question focus [?x]. To show the relation to SPARQL queries, this pseudo query could also be formulated as follows:

```
SELECT ?x WHERE {
   ?x "author" "Deception␣Point" .
   ?x a "author" .
}
```

Listing 2.2: Pseudo query for the question *Who is the author of "Deception Point"?*

To show the necessity of a question focus, consider the question *Who publishes books written by Dan Brown?*. The resulting pseudo query would be

```
SELECT ?x WHERE {
   ?x "publish" ?y .
   ?y "write" "Dan␣Brown" .
   ?x a Person .
   ?y a "book" .
}
```

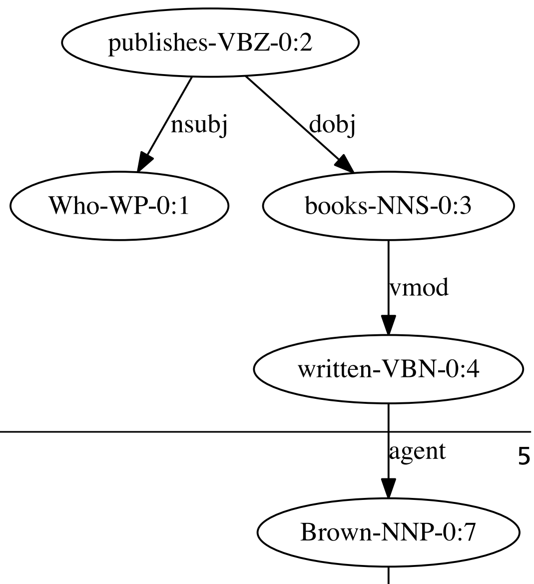
Listing 2.3: Pseudo query for the question *Who publishes books written by Dan Brown?*

As you can see, the query contains multiple variables, of which only one contains the value(s) for the final answer.

2.3.1 Using Stanford dependencies to produce query triples

To construct a triple form of a question as required by a SPARQL query, we need to find out mainly two things: (1) how words from the question relate to each other and (2) which words are important (i.e. *key words*). Hakimov proposed using a dependency tree of words from the question, as produced by the Stanford CoreNLP (SCNLP) [6] library. See figure 2.3.1 for the dependency tree of the question *Who publishes books written by Dan Brown?*. This tree already constains everything we need to know to produce a triple form of the question:

- *publishes* has a subject and an object, we will therefore consider it to be a relation
- *Who* is the subject of this relation
- *books* is the object of this relation
- *books* is further described by an attribute *written*
- The *written* attribute has *Dan Brown* as agent
- *books* is an improper noun ("NNS"), we will therefore consider it to be a variable



- *Dan Brown* is a proper noun ("NNP"), we will therefore consider it not to be a variable

Using these facts, we can construct two triples with the nodes from the dependency tree:

- `[[Who] [publishes] [books]]`
- `[[Dan Brown] [written] [books]]`

Matching and combining triple patterns

In total, there are currently 12 patterns that produce triples from such dependencies. Every pattern may produce an entire triple or just a part of it. For example, the pattern `{agent,comp}(X, Y)` produces the triple `[[?] [X] [Y]]`. In this sentence, this corresponds to the triples `[[?] [publishes] [books]]` as well as `[[?] [written] [Dan Brown]]` (note that `dobj` is a sub-relation of `comp`, see [7]). If an empty slot of a triple matches the arguments of another generated triple, they are merged to form a complete triple.

From triples alone we can not yet produce a valid SPARQL query. In the following I will discuss how the focus variable can be identified and how constraints on the types of variables can be derived.

2.3.2 Identifying the question focus

For question focus identifcation, I chose a simple solution that suffices the style of questions asked in the QALD-2 challenge [6]. The latter served as evaluation, and will be elaborated later in this report. For all 100 questions from the DBpedia training set, the question focus is the first variable (identified as described above) mentioned in the question. Therefore, identifying variables using POS tags is sufficient for finding the question focus in questions from the QALD-2 challenge.

2.3.3 Deriving type constraints

As will be discussed in section 2, it is useful to derive type information for variables in the generated triples. Variables that are question words are directly mapped to RDF types by looking them up in a table:

Who	http://schema.org/Person, http://schema.org/Organization
Where	http://schema.org/Place
When	xsd:date
How many	xsd:integer, xsd:decimal, xsd:double, xsd:float

6 `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=challenge&q=2`

For all other variables, the name of the variable itself (e.g. *book*) is used as indicator for its type. When the pseudo query is mapped to a SPARQL query, the system will attempt to map this indicator to an actual RDF type (e.g. `bibo:Book`) as well.

One of the pseudo queries generated from the dependency tree therefore looks as follows:

```
SELECT ?who WHERE {
   ?who "publish" ?book .
   "Dan␣Brown" "write" ?book .
   ?who a <http://schema.org/Organization> .
   ?book a "book" .
}
```

Listing 2.4: Pseudo query for the question *Who is the author of "Deception Point"?*

All query elements enclosed in quotes will in later steps be mapped to actual URIs. Note that other pseudo queries (with variations of the type constraints) are generated as well, which will be discussed in section 3.

2.4 Mapping pseudo queries to SPARQL queries

The approach I took to map pseudo queries to SPARQL queries is, like Hakimov's approach, based on WordNet [3]. However I chose to not produce any explicit mappings between ontology properties, which Hakimov called *relational patterns*. His approach was to first map predicates to ontology properties based on string similarities, and then to look up further ontology properties that are similar to the one with the highest string similarity. This lookup dictionary (relational patterns) was computed beforehand using a WordNet path similarity threshold. Adjective predicates were previously mapped to their noun representation using WordNet.

This approach has one major problem: It relies on DBpedia's inconsistency regarding object properties to find variations of their lexical representations. This way, for the pseudo triple `[?y "write" "Dan Brown"]`, the object property `dbpedia-owl:author` can only be found because there is another object property called `dbpedia-owl:writer` and there is a path in WordNet between "author" and "writer". Future versions of DBpedia might combine these two properties. At that point, there is no way to map the previously mentioned pseudo triple using only relational patterns.

Using WordNet to find and score lexical variations

Instead of calculating a fixed set of relational patterns, I therefore decided to retrieve lexical variations of a predicate using WordNet in an ad-hoc fashion. For a given predicate `p`, the following relations in WordNet have been utilized. Each relation is associated with a scoring.

- **synonyms**: Words from any of the synsets in which `p` appears. Score is 1.
- **derivationally related words**: Noun or verb forms of `p` if `p` is a verb or noun, resp. Score is 1.
- **transitive synonyms**: Synonyms of synonyms. Score is $1-(depth/maxDepth)$.
- **hyponyms**: Words that imply `p` (e.g. *compose* implies `p`=$create$). Score is $1-(depth/maxDepth)$.

- **hypernyms**: Words that are implied by `p` (e.g. *create* is implied by `p=`*compose*). Score is $0.1-(depth/maxDepth)$.

The same is done to find type constraints where the relevant type is lexically different from the one specified in the question. For example, *movie* may have to be mapped to `dbpedia-owl:Film`. In this case, derivationally related words and hyponyms are not used. The latter would only introduce type constraints that are too specific and will likely lead to wrong results. For example, *book* may be mapped to `dbpedia-owl:Novel` using its hyponyms, which may cause only a subset of the relevant books to be returned.

Generation of *Candidate Queries*

Whenever one of the elements of the original pseudo query is mapped to a specific URI (including properties, resources and types), the results are likely to be ambigous. For example, for "author" there might be the relevant property `dbpedia-owl:author`, but there is also the relevant property `dbpedia-owl:writer`. In this case, all possible interpretations are added as *candidates*. In the end, combinations of all possible candidates are assembled into candidate queries. To be able to sort them by relevante, they are assigned an overall score. The latter is calculated by multiplying scores of all URI matches in the query.

When the system is done generating all candidate queries, they are sorted by their score. The top-scoring query candidate is sent to the SPARQL endpoint, and if it yields any results, it is assumed to be the answer to the question. If not, the second query candidate is tested, and so on, until either all queries are ruled out or an answer is found.

3 A Web Frontend for PAL

To allow for better experimentation and for demonstration purposes, I developed a simple web frontend for PAL. This frontend sends queries entered in the input field to a REST-style Java servlet which responds with an input interpretation and query results in the JSON format. The web frontend then displays a graphical version of the query interpretation along with results to the user. In figure 3 you will find a screenshot of this web frontend.

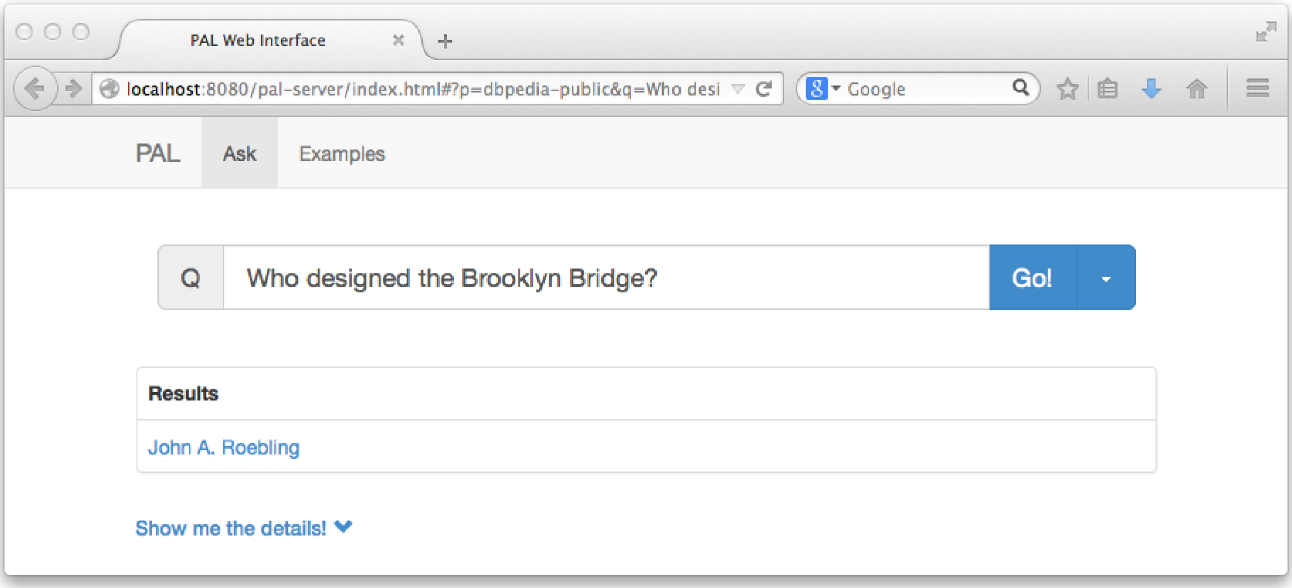


Figure 3.1: Screenshots of the web interface for PAL

How the frontend shows evidence

In figure 3, you can see some of the evidence why PAL assumed the displayed results to be the answer to the entered question. First of all, this includes an *input interpretation*. This is the pseudo query generated from the dependency tree, with all elements in the query triples mapped to the selected ontology (i.e. the ontology provided by the selected SPARQL endpoint). Visible is only the best-scoring interpretation of which the SPARQL query yields at least one result.

If you wish to know more about the elements in the input interpretation, you can click on most of the elements, except for the original lexical forms. Sometimes the URI of a resource does not tell much about itself, and may also not include any human-readable elements. For example, `<http://musicbrainz.org/artist/084308bd-1654-436f-ba03-df6697104e19#_>` represents the band "Green Day" in the MusicBrainz database. In this case, it may be helpful to follow the URL of the resource.

Below the input interpretation, you will see *type constraints* for variables in the query. As for the input interpretation, most elements of the type constraints are clickable. Underneath you will find the final SPARQL query that was send to the SPARQL endpoint to retrieve the displayed results.

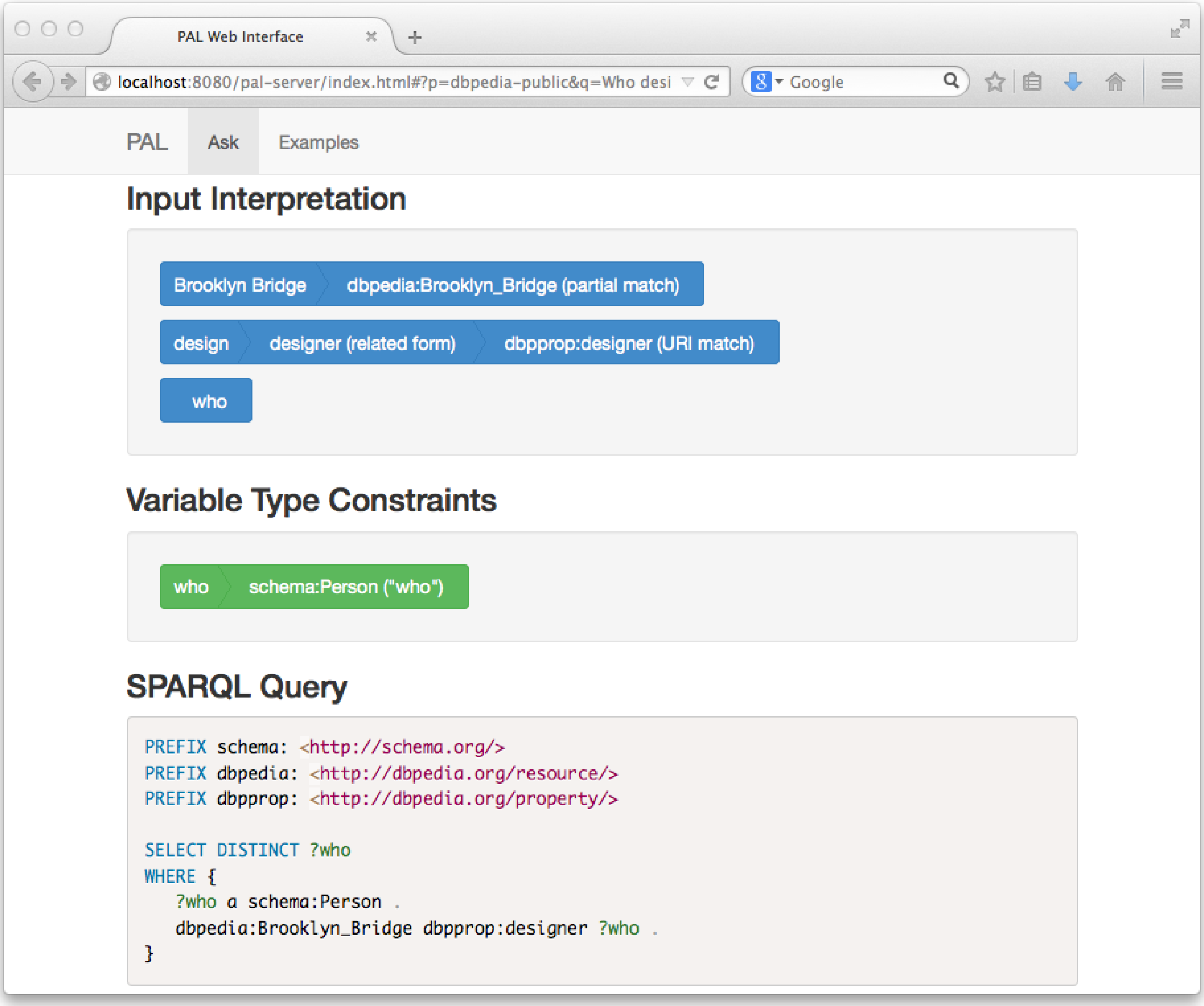


Figure 3.2: Screenshots of the web interface for PAL

4 Evaluation

For my evaluation, I used two question sets from the QALD-2 challenge [1]. All data from the challenge is available on its website, see the link in the footnote. This challenge consists of mostly two main question sets: one for DBpedia, and one for MusicBrainz [2]. Both question sets contain 200 questions each. For both, 100 of these are training questions, and 100 are test questions against which the official evaluation of participating systems was performed.

Results

The following chart shows the evaluation results for the QALD-2 challenge on the DBpedia question sets.

Test set	# Correct	Partially correct	Precision	Recall	F-1 score
Training	27	9	76.8	32.2	45.3
Test	16	5	65.1	20.0	**30.6**

Also listed in the following are the results of other systems that participated in the QALD-2 challenge. However, I was unable to reconstruct the official evaluation results for the participating systems. For example, the best-scoring system "SemSeK" answered 32 out of 100 questions correctly, and 7 partially right, but received a recall of 48%. Since less than 32%+7%=39% of the questions were answered correctly, I cannot reproduce the recall value of 48% using the recall formula as described in the QALD-2 challenge paper.

System	# Correct	Partially correct	Precision	Recall	F-1 score
SemSeK	32	7	44.0	48.0	**46.0**
Alexandria	5	10	43.0	46.0	45.0
MHE	30	12	36.0	40.0	38.0
QAKis	11	4	39.0	37.0	38.0
Hakimov	15	?	>83.3	>15.0	>25.4

[1] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=challenge&q=2

[2] https://musicbrainz.org/

5 Conclusion & Recommendations

5.0.1 What's to be done

As a starting point for what can be done to improve my system, the following list contains problems that came up with the DBpedia training set of questions from the QALD-2 challenge. This list is incomplete, and the numbers are only estimates. For some questions it will be necessary to implement multiple of the missing features, i.e. these features are partly interdependent.

Missing feature	# Qs affected	Comment
Matching of YAGO classes (e.g. `yago:CountriesInEurope`)	14	As of now, e.g. "countries in Europe" is interpreted as a triple, not a type
Erroneous dependency parses	10	In some cases, retrieving the N-best dependency parses might help
Matching of literals	10	Some properties are not linked to URIs, but to literals. As of now, the system assumes everything to be either a variable or a URI.
Combination of multiple possible sparql queries (e.g. for "ruledBy 'SPD'" (literal match) and "ruledBy dbpedia:SPD" (URI match)	10	Likely hard to implement in this system, as it cannot tell when two properties or two property values mean the same.
Yes/No questions	8	May be hard to get right for some "no" answers: The system currently assumes that all URIs involved in the question are connected, which is not the case for a false statement about URIs.
Resource type constraints (starring in the *movie* "Terminator")	8	
Comparison operators (more than, taller than, same as, etc.)	6	
Retrieval of match with highest/lowest value (tallest, earliest, ...)	4	
Counting of results ("how many ...?")	3	Sometimes the answer is a literal value, sometimes the results themselves have to be counted. Solution could be to add both possibilities as candidate queries.
Matching of dates ("born in 1945")	2	Depends on "matching of literals" to be implemented first

6 Related Work

Hakimov et al. [4] proposed a system using what they call *relational patterns*. My work is based on their approach. Details on their system and how it differs from my implementation can be found in section 2.4.

Unger et al. [9], which are also the authors of the QALD challenge, presented an approach that generates SPARQL patterns from the NL query and then attempts to fill the slots. My approach uses patterns as well, but on a more atomic level. While PAL generates triples based on patterns, and then combines them to a pseudo query, Unger's approach generate whole SPARQL candiates based on their patterns. While they are this way able to process more complex queries like *Which cities have more than three Universities?*, which requires the SPARQL query to aggregate and order results, it may require more manual work to produce such patterns.

Ferruci et al. [10] presented a pattern-based approach as well. Like PAL, they used patterns that produce triples (i.e. relations). However, their patterns are of different nature in that they are much more specific to a relation. These patterns are extracted from Wikipedia. This is done by taking the Wikipedia page corresponding to a DBpedia entry, and extracting the first sentence that mentions the arguments of a relation connected to the DBpedia entity. This is then assumed to be a possible lexical representation of this exact relation. All these patterns are then aggregated to form a lexicon of possible lexical representation of every relation in DBpedia.

Aggerwal [5] proposed a multi-lingual approach that leverages heterogeneity of different languages to increase information available to match ontology elements to a NL query. The resulting system is listed as "SemSeK" in the QALD-2 evaluation results and is the highest-scoring participator. As the system I describe in this report, they start by identifying entities and classes matching parts of the NL query, and then match potential relations using Stanford dependencies. To match ontology terms to NL terms, they look up not only semantically similar, but also semantically related terms. For example, this allows matching *married to* to `dbpedia-owl:spouse`. Additionally, they use various other similarity measures to match ontology terms to NL terms.

Bibliography

[1] Christian Bizer. DBpedia âĂŞ A Large-scale , Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web*, 1:1–5, 2014.

[2] J. Chu-Carroll, J. Fan, B.K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty. Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*, 56(3.4):6:1–6:12, 2012.

[3] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*, volume 71 of *Language, Speech, and Communication*. MIT Press, 1998.

[4] Sherzod Hakimov, Hakan Tunc, Marlen Akimaliev, and Erdogan Dogdu. Semantic question answering system over linked data using relational patterns. *Proceedings of the Joint EDBT/ICDT 2013 Workshops on - EDBT '13*, page 83, 2013.

[5] David Hutchison and John C Mitchell. The semantic web - ISWC 2012. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *11th International Semantic Web Conference*, page 704. Springer, 2012.

[6] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. Association for Computational Linguistics, 2014.

[7] Marie-catherine De Marneffe and Christopher D Manning. Stanford typed dependencies manual. *20090110 Httpnlp Stanford*, 40(September):1–22, 2010.

[8] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. *W3C Recommendation*, 2009(January):1–106, 2008.

[9] Christina Unger and Lorenz Bühmann. Template-based question answering over RDF data. *Proceedings of the 21st international conference on World Wide Web*, pages 639–648, 2012.

[10] C Wang, A Kalyanpur, and B K Boguraev. Relation extraction and scoring in DeepQA. *IBM Journal of*, 56(3):1–12, 2012.