

# Appendix: TeBaQA - Template-Based Question Answering using Graph-Pattern Isomorphism

Ricardo Usbeck<sup>1,2</sup>[0000-0002-0191-7211], Peter Nancke<sup>3</sup>, Daniel Vollmers<sup>1</sup>, Richa Jalota<sup>1</sup>[0000-0003-1517-6394], and Axel-Cyrille Ngonga-Ngomo<sup>1</sup>[0000-0001-7112-3516]

<sup>1</sup> Data Science Group, Paderborn University, Germany `firstname.lastname@upb.de`

<sup>2</sup> Fraunhofer IAIS, Germany `ricardo.usbeck@iais.fraunhofer.de`

<sup>3</sup> Eccenca GmbH `peter.nancke@eccenca.de`

## 1 Algorithms for Annotation and Permutation

---

**Algorithm 1:** Creation of all permutations of neighbouring words.

---

**Data:** A list of *words*  
**Result:** A list with all possible combinations of neighbors of a sentence which consist of one or multiple words

```
1 permutations =  $\leftarrow \emptyset$ ;  
2 for  $i = 0$  to  $words.size()$  do  
3   for  $y = 1$  to  $words.size() - 1$  do  
4     if  $y - i < 5$  then  
5       permutation  $\leftarrow words.subList(i, i + y)$ ;  
6       add permutation to permutations;  
7     end  
8   end  
9 end  
10 return permutations
```

---

---

**Algorithm 2:** Annotation of a question with entities, predicates and classes.

---

**Data:** The *question*, the DBpedia *ontology*, a boolean *use\_synonyms* if synonyms shall be used, a preprocessed mapping *hypernyms* of words and their hypernym

**Result:** A list with all found DBpedia entities, classes and properties

```
1 dbpedia_entries ← all found entities in the question from the Spotlight
  annotation;
2 permutations ← create permutations for every successive wordgroup in the
  question;
3 foreach permutation in permutations do
4   if permutation occurs in fulltext description then
5     if levenshtein_ratio(permutation, fulltext_occurrence) ≥ 0.2 then
6       | add related entity to dbpedia_entries;
7     end
8   end
9 end
10 foreach word in question do
11   class_candidate ← generate class URI from word;
12   if class_candidate ∈ classes from ontology then
13     | add class_candidate to dbpedia_entries;
14   end
15   property_candidate ← generate property URI from word;
16   if property_candidate ∈ properties from ontology then
17     | add property_candidate to dbpedia_entries;
18   end
19   hypernym_class_candidate ← generate class URI from the hypernym of the
    word;
20   if hypernym_class_candidate ∈ classes from ontology then
21     | add hypernym_class_candidate to dbpedia_entries;
22   end
23   hypernym_property_candidate ← generate property URI from the hypernym
    of the word;
24   if hypernym_property_candidate ∈ properties from ontology then
25     | add hypernym_property_candidate to dbpedia_entries;
26   end
27 end
28 return dbpedia_entries
```

---

## 2 Determining the Span of Words for Generating Permutations

To determine this limit, 10,000 randomly selected DBpedia entities (Version 2016-10) were examined based on the number of words their label contains. The individual words of an entity are separated in their URI by an underscore (\_). The number of words of an entity can be deducted from the number of underscores + 1. Special characters, e.g., opening or closing brackets, were ignored. Figure 1 shows that 98,46% of the entities in DBpedia consist of 6 or less words.

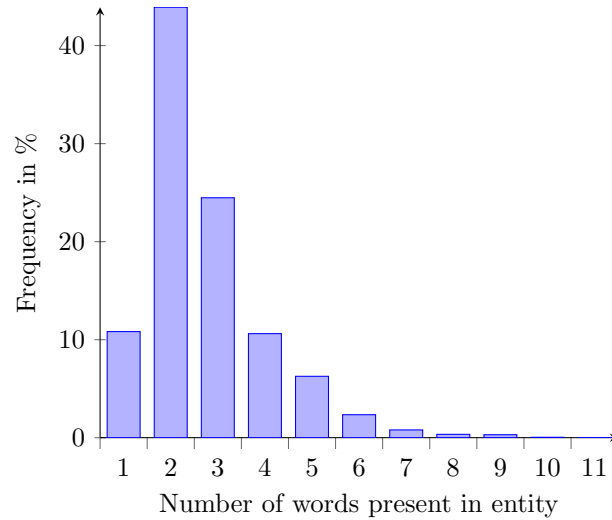


Fig. 1: Distribution of the number of words in 10,000 DBpedia entities.

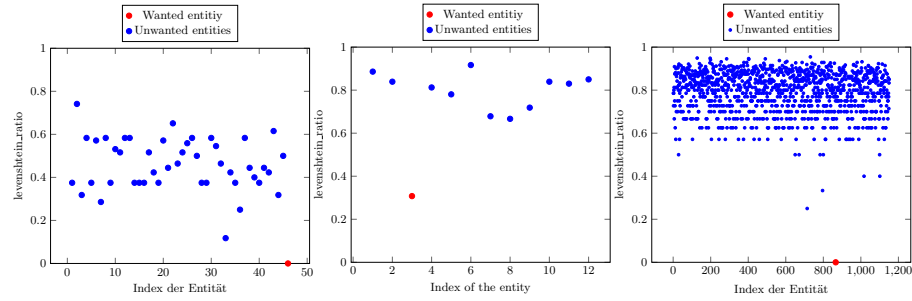
### 2.1 Setting the Threshold value

In order to determine a suitable threshold for the *levenshtein\_ratio*, three diagrams in Figure 2 are examined. They represent all results of the full text search, based on their *levenshtein\_ratio* value in relation to the *searchterm*.

Figure 2a shows *levenshtein\_ratio* values of the entities in which the search term *Game of Thrones* occurs. The red dot represents the desired entity [http://dbpedia.org/resource/Game\\_of\\_Thrones](http://dbpedia.org/resource/Game_of_Thrones) with the *levenshtein\_ratio* of 0. Most entities have a *levenshtein\_ratio* value in the range between 0.2 and 0.6.

Figure 2b represents the *levenshtein\_ratios* of the results for the search term *Exploited*. The searched entity here is [http://dbpedia.org/resource/The\\_Exploited](http://dbpedia.org/resource/The_Exploited) with a *levenshtein\_ratio* of  $\approx 0,31$ . The other entities are between 0,6 and 0,8.

In Figure 2c, 1151 results are found using the search term *Cat*. Except for <http://dbpedia.org/resource/Cat>, most entities have a *levenshtein\_ratio* of 0,5 or higher.



(a) Levenshtein\_ratio for entities of the search term *Game of Thrones*. (b) Levenshtein\_ratio for entities of the search term *Exploited*. (c) Levenshtein\_ratio for entities of the search term *Cat*.

Fig. 2: Levenshtein\_ratio for entities of different search terms

The three diagrams above, show that most of the *levenshtein\_ratios* of the unwanted results are between 0.2 and 1.0. For this reason, the threshold for this value is applied with 0.2. Thus, entities are found which are either identical to the search term or a slight modification of it. These include search terms with spelling mistakes, omitted accents or word groups where words such as *the* or *a* have been omitted.

## 2.2 WEKA Algorithms Evaluation

In Table 1, the algorithms available in WEKA are evaluated using the test dataset of QALD-8 and listed in descending order according to their macro-weighted F-Measure.

Table 1: Evaluation of the ML algorithms of WEKA for the QALD-8 test dataset.

<b>Algorithm</b>	<b>F-Measure</b>	<b>Classification rate</b>
LogitBoost	0.570977	0.741176
MultilayerPerceptron	0.567447	0.901961
IterativeClassifierOptimizer	0.555224	0.678431
KStar	0.551202	0.917647
RandomCommittee	0.549233	0.921569
IBk	0.539949	0.909804
MultiClassClassifier	0.539783	0.72549
ClassificationViaRegression	0.534355	0.654902
RandomizableFilteredClassifier	0.532281	0.913725
Logistic	0.531654	0.705882
PART	0.529683	0.737255
NaiveBayes	0.524158	0.654902
NaiveBayesUpdateable	0.524158	0.654902
OneR	0.506121	0.619608
J48	0.504934	0.666667
Bagging	0.502441	0.682353
BayesNet	0.499273	0.647059
LMT	0.497622	0.603922
SimpleLogistic	0.497622	0.603922
SMO	0.494589	0.666667
LWL	0.487677	0.666667
REPTree	0.474129	0.623529
AdaBoostM1	0.472614	0.603922
DecisionStump	0.472614	0.603922
DecisionTable	0.469675	0.611765
MultiClassClassifierUpdateable	0.467166	0.666667
FilteredClassifier	0.45934	0.647059
JRip	0.437337	0.611765
RandomSubSpace	0.428732	0.615686
CVParameterSelection	0.426293	0.580392
HoeffdingTree	0.426293	0.580392
MultiScheme	0.426293	0.580392
NaiveBayesMultinomialText	0.426293	0.580392
Stacking	0.426293	0.580392
Vote	0.426293	0.580392
ZeroR	0.426293	0.580392

### 2.3 External interfaces

There are several ways to access TeBaQA's Question Answering System. These interfaces are documented below.

**REST interfaces** The first interface is provided for the GERBIL QA benchmark and has the following features:

**HTTP-Path:** /qa/

**HTTP-Method:** POST

**Parameter:** query

**Task:** Answers a question (*query*). The answer is a SPARQL object serialized in JSON.<sup>4</sup>

The answer to the question *Where is Angela Merkel born?* is formatted as follows:

---

```
{
  "questions": [
    {
      "question": {
        "answers": {
          "head": {
            "vars": [
              "x"
            ]
          }
        },
        "results": {
          "bindings": [
            {
              "x": {
                "type": "uri",
                "value": "http://dbpedia.org/resource/Hamburg"
              }
            },
            {
              "x": {
                "type": "uri",
                "value": "http://dbpedia.org/resource/Barmbek-Nord"
              }
            },
            ...
          ]
        }
      }
    ]
  }
}
```

---

<sup>4</sup> see: <https://www.w3.org/TR/sparql11-results-json/>

The second interface is intended for communication between the graphical user interface and the server. It answers the question in the same way as the previous interface, but differs in the format of the answer.

**HTTP-path:** /qa-simple/

**HTTP-method:** POST

**Parameter:** query

**Task:** Answers a question (*query*). The answer is returned as a simple JSON object containing a JSON array of all the answers.

In answer to the question *Where is Angela Merkel born?* this interface would return the following:

---

```
{
  "answers": [
    "http://dbpedia.org/resource/Hamburg",
    "http://dbpedia.org/resource/Barmbek-Nord"
  ]
}
```

---

The third interface is intended exclusively for the graphical user interface. It creates a JSON object which contains the title, the description, the abstract, an image, the link to Wikipedia and DBpedia entity, if this information is available for the requested entity.

**HTTP-path:** /infobox/

**HTTP-method:** GET

**Parameter:** resource

**Task:** Collects selected information about an entity. (*resource*). A JSON object is returned.

The answer is a query with the entity [http://dbpedia.org/resource/Max\\_Horkheimer](http://dbpedia.org/resource/Max_Horkheimer) the following JSON object is returned:

---

```
{
  "messageData": {
    "title": "Max Horkheimer",
    "description": "German philosopher and sociologist",
    "abstract": "Max Horkheimer was a German philosopher and
sociologist who was famous for his work in critical theory
as a member of the 'Frankfurt School' of social research.
Horkheimer addressed authoritarianism, militarism,
economic disruption, environmental crisis, and the poverty
of mass culture using the philosophy of history as a
framework. This became the foundation of critical theory.
His most important works include The Eclipse of Reason (19
47), Between Philosophy and Social Science (1930-1938) and
, in collaboration with Theodor Adorno, The Dialectic of
Enlightenment (1947). Through the Frankfurt School,
Horkheimer planned, supported and made other significant
works possible.",
```

```

"image": "http://commons.wikimedia.org/wiki/Special:FilePath/
  Max_Horkheimer.jpg?width=300",
"buttons": [
  {
    "title": "View in Wikipedia",
    "buttonType": "link",
    "uri": "http://en.wikipedia.org/wiki/Max_Horkheimer",
    "slackStyle": "default"
  }, {
    "title": "View in DBpedia",
    "buttonType": "link",
    "uri": "http://dbpedia.org/resource/Max_Horkheimer",
    "slackStyle": "default"
  }
]
}
}

```

---

## 2.4 Graphic Interface

Figure 3 shows the graphical interface after answering the question *From who was Adorno influenced by?*. In the upper part of the interface there is an input line. Below are links to several sample questions. Under the heading *Answer(s)*: all found answers are listed in the form of tiles. If the answer is an entity, this tile contains all information that the `/infobox/` interface returns about this entity. A response can be an entity, a string, a date, a logical value or a number. In addition, the user can display the SPARQL query with which the answer was found by clicking the *SHOW SPARQ QUERY* button. Bootstrap 3.3.7<sup>5</sup> was used for styling. This is a modular CSS framework published under an MIT license<sup>6</sup>.

<sup>5</sup> <https://getbootstrap.com/docs/3.3/>

<sup>6</sup> <https://github.com/twbs/bootstrap/blob/master/LICENSE>



## TeBaQA

Template Based Question Answering


From who was Adorno influenced by?

Answer

EXAMPLE 1   EXAMPLE 2   EXAMPLE 3   EXAMPLE 4   EXAMPLE 5   EXAMPLE 6   EXAMPLE 7   EXAMPLE 8


Answer(s):  
[SHOW SPARQL QUERY](#)

SELECT DISTINCT \* WHERE { <http://dbpedia.org/resource/Theodor\_W\_Adorno>  
 <http://dbpedia.org/ontology/influencedBy> ?uri . }




**Edmund Husserl**  
**German philosopher, known as the father of phenomenology**  
 Edmund Gustav Albrecht Husserl was a German philosopher who established the school of phenomenology. In his early work, he elaborated critiques of historicism and of psychologism in logic based on ana...

[VIEW IN WIKIPEDIA](#)
[VIEW IN DBPEDIA](#)




**Franz Kafka**  
**Austria-Hungary-Czechoslovakian author**  
 Franz Kafka was a German-language writer of novels and short stories who is widely regarded as one of the major figures of 20th-century literature. His work, which fuses elements of realism and the fa...

[VIEW IN WIKIPEDIA](#)
[VIEW IN DBPEDIA](#)




**Georg Simmel**  
**German sociologist, philosopher, and critic**  
 Georg Simmel was a German sociologist, philosopher, and critic. Simmel was one of the first generation of German sociologists: his neo-Kantian approach laid the foundations for sociological antiposit...

[VIEW IN WIKIPEDIA](#)
[VIEW IN DBPEDIA](#)




**Max Horkheimer**  
**German philosopher and sociologist**  
 Max Horkheimer was a German philosopher and sociologist who was famous for his work in critical theory as a member of the Frankfurt School of social research. Horkheimer addressed authoritarianism,...

[VIEW IN WIKIPEDIA](#)
[VIEW IN DBPEDIA](#)



**Arnold Schoenberg**  
**Austrian composer and painter**  
 Arnold Schoenberg or Schönberg was a Jewish Austrian composer, music theorist, and painter. He was associated with the expressionist movement in German poetry and art, and leader of the Second Viennes...

[VIEW IN WIKIPEDIA](#)
[VIEW IN DBPEDIA](#)



**Max Weber**  
**German sociologist, philosopher, and political economist**  
 Karl Emil Maximilian "Max" Weber was a German sociologist, philosopher, jurist, and political economist whose ideas profoundly influenced social theory and social research. Weber is often cited, wit...

[VIEW IN WIKIPEDIA](#)
[VIEW IN DBPEDIA](#)

Fig. 3: Grafical User Interface.

### 3 Caching

Below, we provide three diagrams of caching times. The average time to classify a question without caching is 0.2497s. The duration of the classification of a question is almost constant for all questions. The average time to annotate and create queries is 2.7691s with a much higher standard deviation of 2.6445.

To determine how much time can ideally be saved by putting all permutations into the cache, a second time measurement was performed with the QALD-8 test questions. In contrast to the first measurement, all permutations occurring in the questions are already in the cache. It is shown in Figure 5. To make the diagrams in Figure 4 and 5 optically comparable, 17s were chosen as the maximum value for the Y-axis.

The third diagram shows a realistic scenario, where the annotated permutation fragments of the questions from the QALD-8-Train dataset were stored in the cache and the time measurement for the QALD-8-Test dataset was performed.

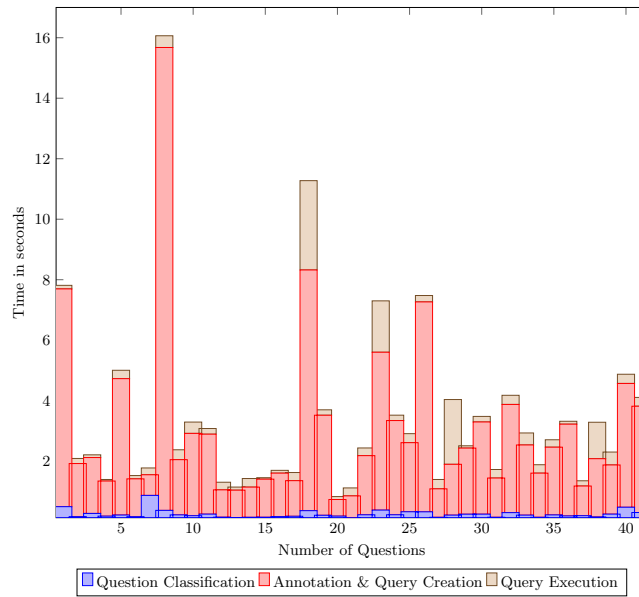


Fig. 4: Time measurement for QALD-8 test data set questions.

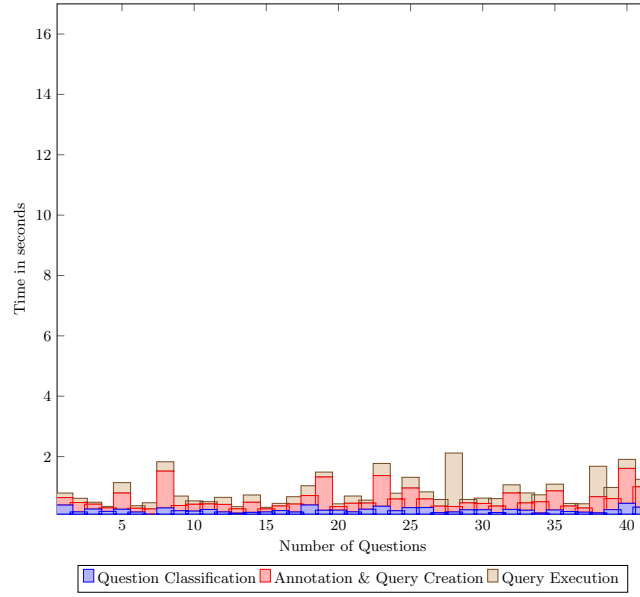


Fig. 5: Time measurement for QALD-8-test with all questions in cache.

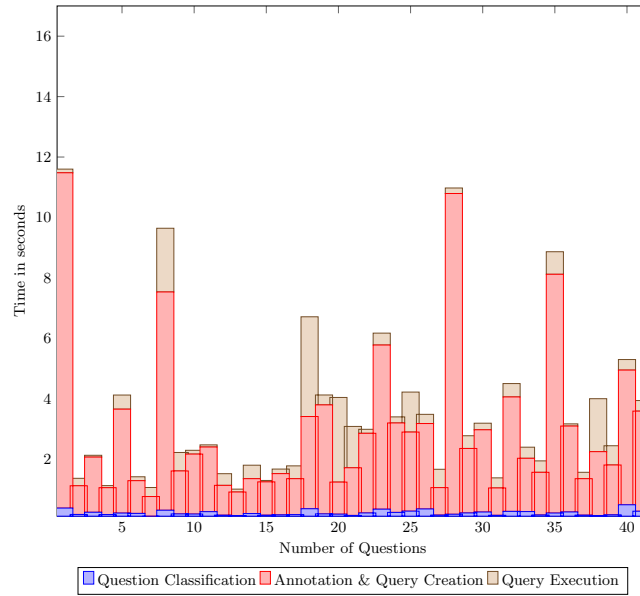


Fig. 6: Time measurement for QALD-8-test with QALD-8-train in cache.