A Large Database of Hypernymy Relations Extracted from the Web

Julian Seitner[1], **Christian Bizer**[1], **Kai Eckert**[2], **Stefano Faralli**[1], **Robert Meusel**[1], **Heiko Paulheim**[1], **Simone Ponzetto**[1]
[1]Data and Web Science Group, University of Mannheim, Germany
jseitner@mail.uni-mannheim.de,{chris,stefano,robert,heiko,simone}@informatik.uni-mannheim.de
[2]Stuttgart Media University
eckert@hdm-stuttgart.de

Abstract

Hypernymy relations (those where an hyponym term shares a "isa" relationship with his hypernym) play a key role for many *Natural Language Processing* (NLP) tasks, e.g. ontology learning, automatically building or extending knowledge bases, or word sense disambiguation and induction. In fact, such relations may provide the basis for the construction of more complex structures such as taxonomies, or be used as effective background knowledge for many word understanding applications. We present a publicly available database containing more than 400 million hypernymy relations we extracted from the CommonCrawl web corpus. We describe the infrastructure we developed to iterate over the web corpus for extracting the hypernymy relations and store them effectively into a large database. This collection of relations represents a rich source of knowledge and may be useful for many researchers. We offer the tuple dataset for public download and an Application Programming Interface (API) to help other researchers programmatically query the database.

Keywords: Hearst patterns, hypernym extraction, big data, Web crawling, Application Programming Interface.

1. Introduction

Hypernymy relations are important in many Artificial Intelligence (AI) and Natural Language Processing (NLP) applications, and in general in data-intensive applications processing large amounts of unstructured and semi-structured information in order to interpret data and text that are not already semantically annotated, assigning types to entities (be it named entities in text, or entities in semi-structured data) represents a crucial step to understanding the data. For instance, in Paulheim and Fürnkranz (2012), it has been shown that adding precise types of instances (in that case taken from Semantic Web resources such as DBpedia and YAGO) to a data mining problem can lead to a significantly improved performance in many data mining tasks. Also when performing data integration, e.g., of a large collection of tabular datasets, into a large knowledge base, first understanding whether the entities in a table are, e.g., cities, states, or mountains, is a very important step towards a high quality integration result (Ritze et al., 2015).

While there are quite a few named entity recognition and disambiguation tools that do serve that purpose and exploit knowledge bases such as Wikipedia, DBpedia, or Freebase, a common problem is dealing with the long tail of less popular entities that are not contained in such knowledge bases. Most of the existing wide-coverage knowledge bases have no problems in identifying, e.g., major cities ("New York is a city") or celebrities ("Madonna is a singer"), but they show limitations with respect to small villages and less known people. However, the potential of web-scale intelligent, data-intensive applications can only be unlocked if they are capable of dealing with the most prominent entities, as well as the long tail. Hence, the necessity of extending existing knowledge bases with hypernymy relations that cover also the long tail entities.

In this paper we present a novel, open resource consisting of more than 400 million tuples extracted from the CommonCrawl[1], each also containing a rich set of attributes such as: the set of patterns matching the pair, the set of the pay-level domains on which the patterns were matched, etc. We released both the dataset and an API (the two can be downloaded at `http://webdatacommons.org/isadb/`) to let other researchers work at large scale in many NLP task (restricted not only on ontology learning).
The rest of this paper is structured as follows. In Section 3., the process of creating the database is described. Section 4. describes bot the released dataset and Java API to programmatically access to the tuples database; and in Section 5., we trace some conclusions about the released resource/API and we discuss about the impact on potential new applications in the field of text understanding.

2. Related work

In the past, many different methods have been developed for hypernym extraction, ranging from simple lexical patterns (Hearst, 1992; Oakes, 2005) to statistical and machine learning techniques (Dolan et al., 1993; Caraballo, 1999; Agirre et al., 2000; Ritter et al., 2009), to name a few.
Snow et al. (2004) first search sentences that contain two terms which are known to be in a taxonomic relation (term pairs are taken from WordNet (Miller et al., 1990)), then parse the sentences, and automatically learn patterns from the parse trees. Finally, they train a hypernym classifier based on these features. Lexico-syntactic patterns are generated for each sentence relating a term to its hypernym, and a dependency parser is used to represent them.
For the ontology learning task, Velardi et al. (2013) induce taxonomies from scratch by extracting hypernyms from a domain corpus and the Web. Definitional sentences such as *"lion is a dangerous animal"* (where *"animal"* is the type of *"lion"*) are recognized by the Word Class Lattices classifier (Navigli and Velardi, 2010) trained on a large set of Wikipedia definitions.

[1]`https://commoncrawl.org`

Kozareva and Hovy (2010) induce a taxonomy using a particular kind of Hearst-like (Hearst, 1992) lexico-syntactic patterns, i.e. so-called Doubly Anchored Patterns (DAP). The hypernymy relations extraction consist of two phases. First the authors bootstrap the terminology harvesting with DAP of the kind *"animals such as lions and *"*, so it is possible to discover new terms such as *"cats"*. Next, for each pair of terms in the discovered terminology e.g. *("lions","cats")* they automatically create a DAP^{-1} of the kind *"* such as lions and cats"* and discover new hypernyms e.g. *"felines"*. The above mentioned works focus on domain-specific hypernymy relations extractions and due to their need of domain constraints - a specific defined term in (Navigli and Velardi, 2010) or a seed pair (Kozareva and Hovy, 2010) - they may not be used to collect the whole set of hyponym-hypernym pairs from a large scale corpus such as the Web.
The *Linked Hypernym Dataset* (Kliegr, 2015) is similar to the work presented in this paper. The authors try to add missing types to DBpedia, which is a common task in knowledge graph completion (Paulheim, 2016). The dataset contains types for DBpedia entities that have been extracted from the corresponding Wikipedia articles using Hearst patterns. In contrast, the work presented in this paper uses the whole Web as a corpus, not only Wikipedia, and hence, it is not limited to find hypernymy relations to entities that are represented by a Wikipedia page, but between arbitrary entities.
Microsoft's *Probase* (Song et al., 2011) is the work closest to ours, albeit not freely accessible. The authors used Hearst-like lexico-syntactic patterns to extract hypernymy relations from 1.68 billion web pages in Microsoft Bing's web corpus, instead of focusing on domain specific hypernymy relations. Probase's main purpose was to create a universal taxonomy containing more than 2.7 million concepts. To this end, the methods underlying Probase are able to extract approximately 25 million pairs.

3. Hypernymy relations extraction

In this section we describe the methodology we applied to extract hypernymy relations from the Web.

3.1. Relation representation

We designed and modeled an hypernymy relation as a tuple, where different attributes related to each extracted "isa" relation can be stored (e.g. statistics, provenance etc.). A tuple T (with reference to Table 1, where we show a real tuple from our dataset) which in wath follow is our data structure which store an hypernymy relation, is defined as $(t_T = (l_t, t, r_t), h_T = (l_h, h, r_h), P_T, S_T, U_T, fr_T, pid_T, pld_T)$ where:

- l_t, t, r_t are the elements of the hyponym component of the *"isa"* relation. Specifically, l_t is the hyponym premodifier, t is the hyponym head noun and r_t is the hyponym post-modifier, e.g. $l_t =$*"second"*, $t =$*"law"* and $r_t =$*"of thermodynamics"*;
- l_h, h, r_h are the elements of the hypernym component of the *"isa"* relation. Specifically, l_h is the hypernymy premodifier, h is the hypernymy head noun and r_h is the hypernymy post-modifier, e.g. $l_h =$*"basic"*, $h =$*"law"* and $r_h =$*"of physics"*;

Table 1: A tuple from our database.

$Tuple\ ID = 12,265,628$					
t_T					
$l_t =$	second	$t =$	law	$r_t =$	of thermodynamics
h_T					
$l_h =$	basic	$h =$	law	$r_h =$	of physic
P_T					
NP, one of the NP					
NP such as NP					
U_TXS_T					
vedicsciences.net					
"This would seem to involve violations of certain basic laws of physics such as conservation of energy, the second law of thermodynamics, and statistical laws of quantum mechanics."					
evolutiondeceit.com					
"The second law of thermodynamics, one of the most basic laws of physics, is based on a very large number of observations and experiments."					
darwinism-watch.com					
"The second law of thermodynamics, one of the most basic laws of physics, is based on a very large number of observations and experiments."					
harunyahya.com					
"The second law of thermodynamics, one of the most basic laws of physics, is based on a very large number of observations and experiments."					
$fr_T =$	4	$pid_T =$	2	$pld_T =$	4

- P_T is the set of patterns matching the hyponym-hypernym pair, that is ($P_T = \{$"NP, one of the NP", "NP such as NP"$\}$) in our example;
- S_T is list of sentences from where the extraction was performed. Due to the nature of the Web, we can have duplicate sentences. In our example, S_T contains a single instance of the sentence *"This would seem to involve violations of certain basic laws of physics such as conservation of energy, the second law of thermodynamics, and statistical laws of quantum mechanics."* and three instances of *"The second law of thermodynamics, one of the most basic laws of physics, is based on a very large number of observations and experiments."*;
- U_T is the set of pay-level domains from where the pair comes, that is, the patterns' provenance (in our example $U_T = \{$`vedicsciences.net`, `evolutiondeceit.com`, `darwinism-watch.com`, `harunyahya.com` $\}$);
- fr_T is the absolute number of hyponym-hypernym pair occurrences (in our example $fr_T = 4$);
- pid_T**:** is equal to $|P_T|$ (in our example $pid_T = 2$);
- pld_T**:** is equal to $|U_T|$ (in our example $pld_T = 4$)[2].

3.2. Extraction workflow

We depict our approach in Figure 1, which consists of three main modules, and corresponding tasks:

1. *WebDataCommons framework:* a framework to access the CommonCrawl dataset;

[2]Note that a pair may occur multiple times within the same Web document.

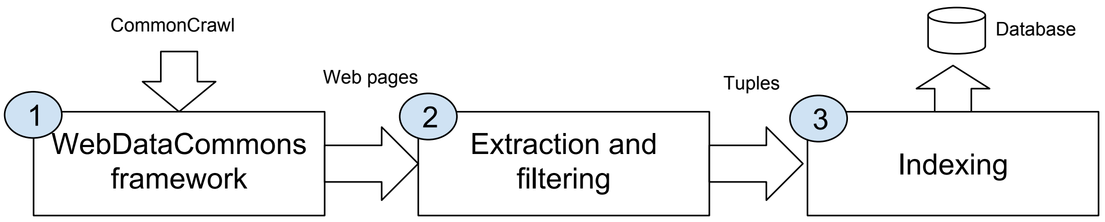

Figure 1: Overview of our method for the tuples extraction.

2. *Extraction and filtering:* is the component of the framework which given a Web page, searches for pattern matches (see Section 3.4.) and generate tuples;

3. *Indexing:* to easily access the large set of extracted tuples, we populated a MongoDB database also released together with a Java Application Programming Interface (see Sections 4.1. and 4.2.).

3.3. WebDataCommons Framework

In order to efficiently parse all websites, we used the framework of the WebDataCommons project[3], which was already used to extract one of the largest hyperlink graphs (Meusel et al., 2015) from the crawl corpora provided by the CommonCrawl Foundation[4] (see Figure 1 block 1), as well as corpora of harvested structured data annotations (Meusel et al., 2014) and HTML tables with relational content (Lehmberg et al., 2015).
The framework is written in Java and tailored towards the Amazon Web Service (AWS) environment. The frameworks allows an efficient parsing of large document collections, like crawl corpora. This workflow is described in Figure 2. In general (1) a queue (SQS) is filled with the references to all documents which needs to be processed. Then (2) a number of servers is requested and is started automatically, which perform all the same process on all available cores:

(A) ask the queue for the next file;

(B) download the file to the server;

(C) process the file using a custom parsing method;

(D) the output is written back to the storage (in the default case S3);

(E) the queue is notified that the file is parsed and can be removed from the queue.

After the queue is empty, (3) the results can be collected from S3. In this process only three actions need to be triggered manually via a command line interface, the remaining actions, including the communication between the different components, is done automatically.
The framework is designed to process a large amount of documents in parallel, but does not allow a communication between the servers, which means that in our case the sorting, aggregation, and cleaning of the extracted tuples needs to be done in a post-processing step, and cannot be done directly within the framework. The original corpus contains over 2.1 billion crawled web pages, consisting of over $38,000$ WARC[5] files with a total packed size of 168TB. The corpus is provided by the Common Crawl Foundation on AWS S3 as free download.[6] The extraction of the tuples took around $2,200$ computing hours and was realized using 100 servers in parallel in less than 24 hours. The next section describes with details the tuple extraction approach, the implementation of which is also avaliable as source code at `http://webdatacommons.org/isadb/`) and can be used to repeat the tuple extraction for different or newer Common Crawl releases.

3.4. Tuple Extraction approach

The tuple extraction is a complex phase where preliminary methodology design decisions have a great impact on the resulting output. We divided this section into three main subsections, each describing few important aspects of the methodology.

Lexico-syntactic patterns. To extract tuples (see Figure 1, block 2) a number of Hearst-like (Hearst, 1992) lexico-syntactic patterns are used. Frequent lexico-syntactic patterns are easily recognizable and indisputably indicate the lexical relation of interest. In particular we focused on 59 patterns that we collected from the past literature. In Table 2 we show the full list of patterns we used for the extraction phase. Eight patterns come from Ponzetto and Strube (2011), where *isa* patterns were used to induce a taxonomy from Wikipedia. Other *isa*-patterns were collected from Orna-Montesinos (2011), where patterns for the term "building" were extracted on a set of specialized textbooks in the field of construction engineering. Remaining patterns were finally collected from Klaussner and Zhekova (2011) where the authors extract *isa* relations from selected Wikipedia pages.
The patterns identified in literature are then translated into regular expressions. For example, (with respect to Table 2) the pattern $p5$ (i.e. "NP_h such as NP_t" where NP_t indicates the hyponym and NP_h the hypernym) was translated into the following regular expression:

```
(\p{L}|\d)[\"\']?\,?\ssuch\sas\s[\'\"]?(\p{L}|\d)
```

As the corpus is read line by line, the length of line has an impact on the performance of a regular expression. Lines

[3]`http://webdatacommons.org/framework/`

[4]`http://commoncrawl.org`

[5]Web ARChive, ISO 28500:2009

[6]`http://blog.commoncrawl.org/2015/05/april-2015-crawl-archive-available/`

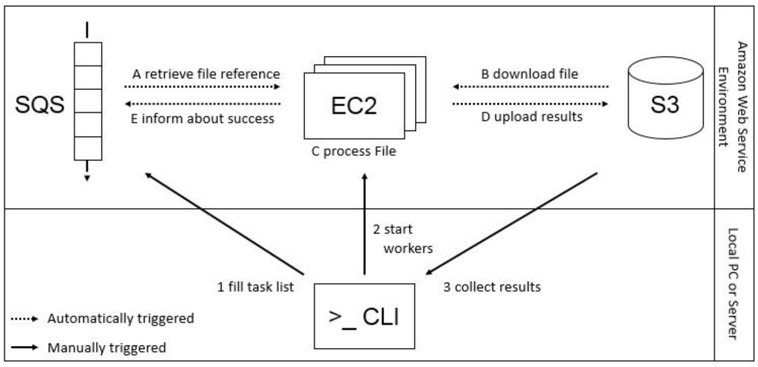

Figure 2: Overview of our workflow of the Web Data Commons Extraction framework.

longer than 100,000 characters are rather frequent in the corpus and therefore the lines have to be split before a regex can be matched. To cope with the line length issue we adopted a sentence splitter strategy; in other words we splitted a line into sentences to analyze each single sentence with regular expressions. For this purpose we defined other regular expressions to split a line of text into sentences.

In order to test the quality of the above defined regular expressions we extracted a random 1% portion of the entire corpus and analyzed 100 matches per pattern. With that evaluation, we estimated the precision of each pattern, as shown in Table 2. Patterns with a very low precision have then been excluded before performing the subsequent steps.

Noun phrase identification: In order to identify noun phrases we used the Stanford POS Tagger (Toutanova et al., 2003) to obtain the mapping of words to lexical categories. The noun phrase identification basically entails a selection of allowed part of speech (POS) tags for the pre/post modifiers and for the head noun.
We defined as head noun allowed tags: singular noun or mass (NN), plural noun (NNS), singular proper noun (NNP), plural proper noun (NNPS). For the premodifier, we added the following tags we selected for the previous set: adjective (JJ), comparative adjective (JJR), superlative adjective (JJS) and past participle verb (VBN).
The allowed tags for the postmodfier include: adjective (JJ), comparative adjective (JJR), superlative adjective (JJS), past participle verb (VBN), gerund or present participle verb (VBG), singular or mass noun (NN), plural noun (NNS), singular proper noun (NNP), plural proper noun (NNPS), preposition or subordinating conjunction (IN), cardinal number (CD) and determiner (DT).

Filtering: Since none of the patterns are perfect (neither is content on the Web), some post-processing and filtering is required to reduce the amount of noise in the database. We try to facilitate a sensible trade-off between coverage and precision. In general, we try to remove only the obvious noise, while keeping as much coverage as possible. This strategy comes from the idea that some task may need “less precise” but “more covering” data. In use cases where more precision is required, we provide metadata for each tuple that allows for additional filtering techniques on the client side. Our filtering strategy may be summarized as follows:

- **Removal of duplicates:** due to the implementation of the crawler used by the CommonCrawl foundation and the nature of content on the Web, the same web page may be replicated in the corpus. To reduce the impact of spam and to decrease the size of the extracted tuples, tuples that occur more than once under the same pay level domain are removed. We also made a duplication removal at sentence level (based on exact match).

- **Tuple normalization:** At this stage we apply a post-processing tuple normalization: i) we transform all the capital letters to lower case and removed all leading and trailing punctuations. ii) we remove all quotation marks and apostrophes, since apostrophes are frequently used as replacement for quotation marks;[7] iii) we analyze different kinds of punctuation occurrences. This step does not only target the optional commas or the commas of co-ordinations, but also full stops, exclamation marks, question marks, colons, and semicolons. These usually occur at the end of a word and are followed by a space. If this is the case these punctuations are removed. However, sentences on the web are not always written with the proper syntax and the space behind a punctuation is simply left out. This has an adverse effect on the POS tagger and consequently the noun phrase extraction. After the removal of interpunctuated symbols we performed a length check. If one entity of a tuple is longer than 50 characters, it is regarded as noise and the tuple is removed from the processing pipeline. This length is the result of empirical tests (a check of 100 tuples with an entity longer than 50 characters, showed that only 8 percent of these were correct); iv) we lemmatized each word of a noun phrase. With the lemmatized entities a further refactoring step is taken, which targets noun phrases that contain multiple nouns. To be more specific, this regards noun phrases, which are located in front of the pattern and contain nouns separated by a preposition (complex NPS). The issue regarding these entities is, that it is not trivial to determine, which of these nouns is in a hyponymy relation with the entity behind the pattern. For example given a pattern “* such as *”:

[7] we preserved the apostrophes in the genitive form of a noun

Table 2: The list of patterns used for the tuples extraction (NP_t indicates the hyponym and NP_h the hypernym).

ID	Pattern	Precision
p1	NP_t and other NP_h	0.70
p2	NP_h especially NP_t	0.19
p3a	NP_h including NP_t	0.44
p4	NP_t or other NP_h	0.70
p5	NP_h such as NP_t	0.58
p6	NP_t and any other NP_h	0.76
p7	NP_t and some other NP_h	0.54
p8	NP_t is a NP_h	0.44
p8b	NP_t was a NP_h	0.39
p8c	NP_t are a NP_h	0.57
p8d	NP_t were a NP_h	0.42
p9	NP_h like NP_t	0.17
p10	such NP_h as NP_t	0.58
p11	NP_t like other NP_h	0.31
p12a	NP_t, one of the NP_h	0.38
p12b	NP_t, one of these NP_h	0.13
p12c	NP_t, one of those NP_h	0.15
p13	examples of NP_h is NP_t	0.33
p14	examples of NP_h are NP_t	0.45
p15a	NP_t are examples of NP_h	0.20
p15b	NP_t is example of NP_h	0.36
p16	NP_h for example NP_t	0.31
p20a	NP_t is adj_{sup} NP_h	0.63
p20b	NP_t are adj_{sup} NP_h	0.41
p20c	NP_t is adj_{sup} most NP_h	0.63
p20d	NP_t are adj_{sup} most NP_h	0.49
p21a	adj_{sup} NP_h is NP_t	0.25
p21b	adj_{sup} most NP_h are NP_t	0.19
p21c	adj_{sup} NP_h is NP_t	0.31
p21d	adj_{sup} most NP_h are NP_t	0.21
p22a	NP_t which is called NP_h	0.50
p22b	NP_t which is named NP_h	0.26
p23a	NP_h mainly NP_t	0.22
p23b	NP_h mostly NP_t	0.16
p23c	NP_h notably NP_t	0.28
p23d	NP_h particularly NP_t	0.19
p23e	NP_h principally NP_t	0.26
p24	NP_h in particular NP_t	0.25
p25	NP_h except NP_t	0.22
p26	NP_h other than NP_t	0.44
p27a	NP_h e.g. NP_t	0.33
p27b	NP_h i.e. NP_t	0.29
p28a	NP_t, a kind of NP_h	0.18
p28b	NP_t, kinds of NP_h	0.45
p28c	NP_t, a form of NP_h	0.18
p28d	NP_t, forms of NP_h	0.33
p29a	NP_t which look like NP_h	0.13
p29c	NP_t which sound like NP_h	0.18
p30a	NP_h which are similar to NP_t	0.28
p30b	NP_h which is similar to NP_t	0.29
p31a	NP_h example of this is NP_t	0.25
p31b	NP_h examples of this are NP_t	0.18
p34	NP_h types NP_t	0.17
p35	NP_t NP_h types	0.12
p36	NP_h whether NP_t or	0.13
p37	compare NP_t with NP_h	0.16
p38	NP_h compared to NP_t	0.17
p39	NP_h among them NP_t	0.23
p40	NP_t as NP_h	0.17
p41	NP_h NP_t for instance	0.13
p42	NP_t or the many NP_h	0.31
p43	NP_t sort of NP_h	0.18
p44	NP_t sort of NP_h	0.14

Table 3: Statistics of the released dataset.

number of used patterns	58
$\|\{T\}\|$	401,150,041
$\|\{t_T\}\|$	120,992,255
$\|\{h_T\}\|$	107,691,822
$\|\{t_T\} \cap \{h_T\}\|$	16,528,348
Avg. patterns matches per tuple	1.1
Avg. pay level domain matches	1.3
Avg. (t_T, h_T) count	1.5
Max. patterns per (t_T, h_T)	52

1) "Works of artists such as Vivaldi", where the last noun is in hyponymy relation with the entity behind the pattern. Consequently it would be wrong to take the first noun as head noun;
2) "Works of artists such as The Four Seasons", where this example depicts the opposite, as it is the first noun that is correct. In this case the noun in the post-modifier is - as intended - a specialization of the first noun;
3) "A variety of works such as The Four Seasons", where it appears to be similar to the first one; However, the leading noun is a so called collective noun, as it refers to a group of entities.

To find resolution for the latter problem, we inspected a random sample of 100 tuples with multiple nouns. Out of those, there were 21 starting with a collective noun. In all of these 21 instances the latter noun was the correct one. Therefore, the decision was made to remove collective nouns and the subsequent preposition from the tuple.

- **Tuple aggregation:** The tuple aggregation is the last step in the pipeline, before the tuples database creation. Two tuples x and y are aggregated if t_x equal to t_y and if h_x equal to h_y, in other words, if the two tuples share the same hyponym and the hypernym head nouns. In this phase we not only reduce the data volume significantly, but also generate three useful statistics: i) the count of the occurrences fr_T of a tuple T; ii) the amount of distinct patterns pid_T, which extracted a tuple T; iii) the number pld_T of pay-level domains from where the pair was extracted.

3.5. Statistics

In Table 3 we show some statistics about the resulting dataset. Thanks to our approach, we are able to extract $401,150,041$ *isa* relations, and identify $120,992,255$ and $107,691,822$ unique hyponyms and hypernyms, respectively. Table 4 shows the 10 most frequent matching patterns. As expected with $242,370,719$ matches the list is dominated by the "NP_t is a NP_h" pattern.
Since there are more than 120M hyponyms, i.e., concepts which we can assign at least one type, the number of entities is by one to two orders of magnitude larger than that of other freely available knowledge graphs, such as DBpedia, YAGO, or Wikidata.[8]

[8] See Paulheim (2016) for comparison figures.

Table 4: Top 10 frequent patterns.

count	pattern
$242,370,719$	NP_t is a NP_h
$80,640,885$	NP_h including NP_t
$70,337,543$	NP_h such as NP_t
$45,900,092$	NP_t and other NP_h
$20,872,227$	NP_h especially NP_t
$13,392,348$	NP_t or other NP_h
$11,656,254$	NP_h (mainly|mostly|notably|particularly|usually |principally|especially) NP_t
$11,080,276$	NP_h types NP_t
$10,360,953$	adj_{sup} NP_h be NP_t
$9,648,662$	NP_h (like|except) NP_t

Table 5: Excerpt from the top $1,000$ frequent (t_T, h_T).

t_T	h_T
worldcat	linked data resources
telephone number	basic information
chainmail	form of spam
video	content
phone number	datum
united states	country
estimate datum	special algorithm
gsview	pdf viewer
english	language
socialcam	video app for iphone
copper	conductive material
social security number	personal information

In Table 5 we show some examples for interesting hypernymy relations from the top occurring on the entire web. Please note how the meaning of "most interesting" may vary across uses, for example some study may be more interested on analyzing domain specific hypernymy relations which are assumed to be in the "long tail" and not between the most frequent.

We also report in Table 6 the 15 pay level domains containing the highest number of extracted hypernymy relations. We noticed that this list contains: i) the most important blog sites (i.e. `blogspot.com`, `wordpress.com`, etc.); ii) Google

Table 6: Top 15 frequent URLs.

count	urls
$32,114,019$	blogspot.com
$31,230,507$	google.com
$11,220,307$	wordpress.com
$4,542,604$	google.es
$3,334,085$	docstoc.com
$3,153,578$	google.fr
$3,015,504$	wikipedia.org
$2,975,284$	google.com.au
$2,752,864$	google.ca
$2,574,356$	faqs.org
$2,490,576$	slideshare.net
$2,488,768$	archive.org
$2,425,906$	issuu.com
$2,230,084$	google.co.uk
$2,184,607$	typepad.com

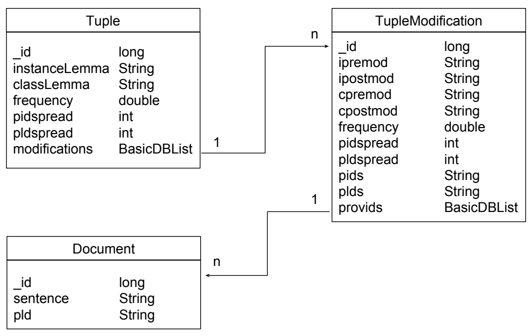

Figure 3: The Tuples database schema.

sites in different languages; iii) reference Web content (e.g., `wikipedia.org`).

4. Resources

We offer the tuple dataset for public download and an Application Programming Interface (API) to help other researchers programmatically query the tuple dataset. The dataset and the API are available at `http://webdatacommons.org/isadb/`. At the same URL it is possible to download:

- the tuple dataset in the form of a MongoDB-based database;
- the tuple dataset in the form of tables, as compressed tab separated values (TSV) files;
- the Java API package, to programmatically access the tuple from the MongoDB dataset;
- the Java implementation of the tuple extraction approach (see Section 3.4.).

A quick guide on how to install the downloaded software and how to use the Java API is also available.

4.1. Tuple MongoDB

To store and access all the extracted tuples we created a database (see Figure 1, block 3). During prototyping we experimented with many database management systems (DBMS) and found MongoDB (Plugge et al., 2010) (a NoSQL DBMS) as the most suitable technology to store our large set of tuples. NoSQL technologies are designed to scale with "big data" and such databases support dynamic schema design, offering the potential for increased flexibility, scalability and customization.

Figure 3 shows the resulting database schema. The design of that schema was guided by the fact that we aimed at favouring the access to hypernymy relations of the kind (t, h), where t and h are head nouns (see Section 3.). The entity *Tuple* defines such basic hypernymy relation and represents the most used background knowledge to many text understanding tasks. Other additional information may be accessed as a list of *TupleModification*. The *TupleModification* entity represents a variant of a tuple where t and h are preserved but the left and right modifiers (i.e. l_t, l_h, r_t, r_h) changes. From each tuple variant we can also access to the list of:

- *Pattern*: the set containing the patterns which matched the “isa” relation;
- *Pay-level domain*: the Urls of the Web documents from where the relation was extracted;
- *Documents*: entities of the kind (pay-level domain, sentence) from which the relation was extracted.

4.2. Java API to access the MongoDB

We next describe the Java Application Programming Interface to programmatically query the database. The class ”TuplesDb“ is the main class from where access the tuples database and can be instantiated as follows:

```
...
TuplesDb tDb=TuplesDb.getInstance();
...
```

To query the database and to iterate through all the tuples, the class ”TuplesDb“ has a public method ”getAllTuples()“ to retrieve an instance of the ”AllTuplesResultIterator“ class, which implements the standard Java ”Iterator“ interface:

```
AllTuplesResultIterator
    qri=tdb.getAllTuples();
while (qri.hasNext()) {
 for (Tuple t:qri.next())
 System.out.println(t);
}
```

The class ”TuplesDb“ has also public methods to express many kinds of queries. All of them return specialized a class instance which implements the standard Java iterator interface. For example, if we want to print all the hypernyms h_T of the tuple T where the instance lemma t_T is equal to ”gaga“:

```
TupleQueryResultIterator tqri=
tdb.getAllTuplesWhereInstanceLemma("gaga");
while (tqri.hasNext()){
 for (Tuple T:tqri.next())
   System.out.println(T.getClassLemma());
}
```

Other queries can be more sophisticated and include:

- thresholds for the minimum and maximum values of the tuple frequency fr_T, number of matching patterns pid_T and number of pay-level domain pld_T attributes;
- a specific set of pattern ids $PIDs$, each resulting tuple T has to include in P_T;
- a specific set of pay-level domain ids, each resulting tuple T has to include in U_T;

5. Conclusion

We presented an overview of a database of *isa* relations automatically extracted from the CommonCrawl, the largest existing repository of Web content. This is, to the best of our knowledge, the largest publicly available resource of hypernymy relations from textual resources. We described the resource with some statistics and the database schema designed for storing the data, as well as the Java API we developed to let programmatically query the tuples database. Our database represents a first step towards a more complex semantic resource such as full-fledged taxonomies. In fact, we already used it as part of a SemEval competition on taxonomy induction (Bordea et al., 2015), where we achieved a competitive performance by leveraging our *isa* relations harvested from the Web (Panchenko et al., 2016).

The WebIsaDatabase and the Java API can be downloaded at `http://webdatacommons.org/isadb/`.

Acknowledgements

This work was partially funded by the Deutsche Forschungsgemeinschaft within the JOIN-T project (research grant PO 1900/1-1). Part of the computational resources used for this work were provide by an Amazon AWS in Education Grant award.

References

Agirre, E., Ansa, O., Hovy, E. H., and Martínez, D. (2000). Enriching very large ontologies using the WWW. In *Proceedings of the ECAI Workshop on Ontology Learning*.

Bordea, G., Buitelaar, P., Faralli, S., and Navigli, R. (2015). Semeval-2015 task 17: Taxonomy extraction evaluation. In *Proceedings of the 9th International Workshop on Semantic Evaluation*.

Caraballo, S. A. (1999). Automatic construction of a hypernym-labeled noun hierarchy from text. In *Proceedings of ACL-99*, pages 120–126.

Dolan, W., Vanderwende, L., and Richardson, S. D. (1993). Automatically deriving structured knowledge bases from on-line dictionaries. In *Proceedings of PACLING-93*, pages 5–14.

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING ’92*, pages 539–545.

Klaussner, C. and Zhekova, D. (2011). Pattern-based ontology construction from selected wikipedia pages. In *Proceedings of the RANLP-11 Student Research Workshop*.

Kliegr, T. (2015). Linked hypernyms: Enriching DBpedia with Targeted Hypernym Discovery. *Web Semantics: Science, Services and Agents on the World Wide Web*, 31(1):59 – 69.

Kozareva, Z. and Hovy, E. (2010). A semi-supervised method to learn and construct taxonomies using the web. In *Proceedings of EMNLP-10*, pages 1110–1118.

Lehmberg, O., Ritze, D., Ristoski, P., Meusel, R., Paulheim, H., and Bizer, C. (2015). The Mannheim Search Join Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 35(3):159 – 166.

Meusel, R., Petrovski, P., and Bizer, C. (2014). The webdatacommons microdata, rdfa and microformat dataset series. In *The semantic Web – ISWC 2014*, pages 277–292. Springer.

Meusel, R., Vigna, S., Lehmberg, O., and Bizer, C. (2015). The Graph Structure in the Web - Analyzed on Different Aggregation Levels. *The Journal of Web Science*, 1(1):33–47.

Miller, G. A., Beckwith, R., Fellbaum, C. D., Gross, D., and Miller, K. (1990). WordNet: an online lexical database. *International Journal of Lexicography*, 3(4):235–244.

Navigli, R. and Velardi, P. (2010). Learning word-class lattices for definition and hypernym extraction. In *Proceedings of ACL-10*, pages 1318–1327.

Oakes, M. P. (2005). Using Hearst's rules for the automatic acquisition of hyponyms for mining a pharmaceutical corpus. In *Proceedings of the RANLP Text Mining Workshop*, pages 63–67.

Orna-Montesinos, C. (2011). Words and patterns: lexicogrammatical patterns and semantic relations in domain-specific discourses. *Revista Alicantina de Estudios Ingleses*, 24:213–233.

Panchenko, A., Faralli, S., Ruppert, E., Remus, S., Naets, H., Fairon, C., Ponzetto, S. P., and Biemann, C. (2016). Taxi: a taxonomy induction method based on lexico-syntactic patterns, substrings and focused crawling. In *Proceedings of the 10th International Workshop on Semantic Evaluation*.

Paulheim, H. and Fürnkranz, J. (2012). Unsupervised Generation of Data Mining Features from Linked Open Data. In *Proceedings of WIMS'12*.

Paulheim, H. (2016). Knowlegde Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web Journal*. To appear.

Plugge, E., Hawkins, T., and Membrey, P. (2010). *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*. Apress, Berkely, Cal.,, 1st edition.

Ponzetto, S. P. and Strube, M. (2011). Taxonomy induction based on a collaboratively built knowledge repository. *Artificial Intelligence*, 175(9-10):1737–1756.

Ritter, A., Soderland, S., and Etzioni, O. (2009). What is this, anyway: Automatic hypernym discovery. In *Proceedings of the 2009 AAAI Spring Symposium on Learning by Reading and Learning to Read*, pages 88–93.

Ritze, D., Lehmberg, O., and Bizer, C. (2015). Matching HTML tables to DBpedia. In *Proceedings of WIMS-15*.

Snow, R., Jurafsky, D., and Ng, A. Y. (2004). Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of NIPS-04*, pages 1297–1304.

Song, Y., Wang, H., Wang, Z., Li, H., and Chen, W. (2011). Short text conceptualization using a probabilistic knowledgebase. In *Proceedings of IJCAI-11*, pages 2330–2336. AAAI Press.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of NAACL-03*, pages 173–180.

Velardi, P., Faralli, S., and Navigli, R. (2013). OntoLearn Reloaded: A Graph-Based Algorithm for Taxonomy Induction. *Computational Linguistics*, 39(3):665–707.