# The Web Tuples Database:
# A Large-scale Resource of hyponymy Relations

Master Thesis

presented by
Julian Felix Maria Seitner
Matriculation Number 1397080

submitted to the
Data and Web Science Group
Prof. Dr. Simone Paolo Ponzetto
University of Mannheim

October 2015

# Contents

# List of Algorithms

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Since the era of Web 2.0 the amounts of publicly available information on the web has increased exponentially. The web has become a place where people can easily exchange information and opinions in natural texts on forums, social networks and blogs. Therefore, to make this information available in structured forms becomes an increasingly important task. For example, to improve the automatic understanding of natural language texts [43] [45]. In [36] such process includes the recognition of semantic classes for noun phrases occurring in the text of interest. The relations between noun phrases and their semantic classes are named hyponymy relations.

Hence, to contribute to this research area, this work has the goal to provide a large scale and easy to integrate data source that contains vast amounts of hyponymy relations extracted from the web. Such resource can be used in many field of natural language processing (NLP) for example to induce taxonomies [33] [45] [15].

The challenge is to create such a data source economically. This includes the quality of the final data source in terms of precision and recall, the duration of the processing pipeline, as well as the costs for infrastructure.

The contribution of this work is four-fold:

1. a system is created that executes the hyponymy extraction from a corpus of 1.68 billion web pages. The extraction resulted in 1.1 billion hyponymy relations. In order to enable a extensive analysis the extracted data has to be transformed;
2. A system that aggregates the extracted tuples and generates features for fur-

ther data analysis. The aggregated data contains over $0.4$ billion unique hyponymy tuples and provides detailed provenance information about it.

3. a comprehensive storage system is provided, which enables a fast access a large-scale resource of hyponymy relations;

4. An API is released for fast data exploration. Researchers and other interested parties are then able to query the extracted relations with a simple specification of search parameters.[1]

[1]The system specifications, descriptions of the functionality and the source code for all of these systems are provided in this thesis.

# Chapter 2

# Problem Statement and Contribution

The main motivation of this work is based on the fact that there are no publicly available resources containing a domain-independent and large-scale dataset of hyponymy relations. This is not surprising as there is a plethora of problems to overcome creating such a web Tuple Database (WTDB). The purpose of the WTDB is to enable researchers and other interested parties to use and evaluate information extracted from a large and heterogeneous corpus [9] [13]. Moreover, to enable more interesting computational uses, for each tuple a set of additional attributes is also extracted. These attributes are generated during the aggregation of the tuples using their information about the provenance. Increasing the variety of provenance information will in turn increase the value ranges for the generated attributes, which allows a more fine grained analysis of extracted information. Thus it is a guiding rule for the extraction to achieve a high recall, in order to increase the variety of tuple provenance. Furthermore a high recall will increase the comprehensiveness of the WTDB.
The mere size of the web corpus of 168 terabytes (TB) cannot be process on a single machine in a reasonable amount of time. This entails the usage of highly scalable extraction techniques [13]. In this thesis the extraction is performed in a distributed environment, which adds complexity to the infrastructure as well as the algorithms. As this infrastructure is also expensive, the extraction has to be performed fast in one single run. Thus, the extraction of tuples has to be fast an robust. This is a difficult task, because the way the data is stored in corpus varies drastically and can not be determined easily before the extraction is started. The extraction of tuples requires the identification and evaluation of suitable patterns to cover the hyponymy relational space. Researchers have extracted hyponymy

relations using textual patterns. However, compared to previous experiments, this work is far more extensive in terms of the analysed corpus, the number of applied patterns and is not restricted to any domain.
The high recall of the extraction in combination with the noisy nature of the corpus is a further problem for the WTDB development. On the one hand there is a magnitude of duplicate information stored on the web, especially in forms of automatically generated content. Such content is quite frequent and does not hold much valuable information. This problem has to be considered in this work, because it affects the attributes that are generated from the tuple provenance. On the other hand, as people are free to express any opinion in any form, it is difficult to aggregate tuples automatically. It is easy for the human eye to distinguish that the terms *THING* and *"thing"* refer or not to the same concept in their respective contexts. However, machines do not. Therefore, it is a challenge to aggregate tuples from such a corpus. The expected high recall of the extraction adds an extra level of complexity to this task. The more tuples are extracted, the more system memory is required for the aggregation. With such a large-scale extraction the infrastructural limitations have to be overcome, with scalable algorithms.

The actual aggregated data itself is a valuable contribution as well for tasks such as taxonomy induction, benefits from the provenance information crystallized into attributes for tuples.

This new data source enables the analysis of tuples in three statistical dimensions. For each unique tuple it is measured:

- how often it occurred in the corpus;
- how many different patterns found the same tuple;
- how many differents pay level domains hosted the extracted relation.

The above mentioned attributes indicate the strength of a relation and can be used by researchers to automatically extract portions of interests of the dataset. As the aggregated data contains more then 0.4 billion tuples, to extract single pieces of information is a time consuming task. To cope with the time costs in this work the data is stored in a database.

On the one hand it is not surprising, that the aggregation of a large scale extraction leads to some really large data objects. On the other hand, the variety of data stored in the corpus cause a tremendous amount of rather small objects. Nevertheless, each object has to be stored in a way that enables fast access, in terms of a fast query execution. This can be achieved using a proper database management system (DBMS). All DBMS have different technical limitations [24]. Thus, a DBMS has to be selected and implemented in a way that the vast amount of stored data, does not exceed the technical limits. Furthermore the creation and implementation of a custom storage concept is required to enable a fast and memory efficient query execution. Storing the aggregated tuples in a commonly used DBMS is an important contribution. The choice of commonly used disk-storage DBMS enables an user-friendly integration of the data into other systems. The widespread usage and an active online community facilitate the ease of integration. Additionally a nested data model is developed that enables the fast retrieval of hyponyms for a given noun phrase, without using any joins.

The source code, a system specification and a description of the functionality are provided in this thesis. This will enable researchers to extract hyponymy relations from different English corpora covering different domains. The developed framework can be of high interest also for companies. In fact a fast extraction of relations from large textual resources can be source of a strategic advantage. Being able to extract structured information within days or even hours is the key to seize such an advantage first. The extracted data is a further asset. The extracted tuples can be used by researchers to analyse the precision and recall of a magnitude of hyponymy patterns on a large and noisy corpus. To improve such an analysis, the tuples are cleaned, aggregated and structured.
Although the database can be set up quickly, a user first has to learn the underlying data model and a query language to retrieve data efficiently. Even with an active and helpful online community, it takes time to write efficient queries. These aspects may prevent a fast and user friendly exploration of the data. Furthermore, an unrestricted query can cause the DBMS to fail and shut down, if a query can not be executed on the given infrastructure. These problems are solved by providing an application programming interface (API). On the one hand the API has to enable the user to specify search parameters to extract detailed information from data-source. On the other hand, the API has to be tailored to the storage concept to guarantee a robust query execution. These contradicting goals - free exploration of data and the robust execution of queries - make the implementation a complex task. The contribution of this API is practical. It is coded in a widely used programming language and tailored to the data model of stored tuples. These characteristics of the API enable practitioners a fast, robust and easy to learn access to the new data source.
For each of the above mentioned problems experiments are carried out and discussed later in this thesis. The related work is discussed in the following chapter.

Chapter 3

# Related Work

This research aims at creating a structured data resource containing hyponymy tuples from a large scale web corpus. Since the web contains texts describing any thinkable domain, this research is related to works in the area of Open Information Extraction (Open IE):. “Open IE is an unsupervised strategy to draw out relations from text without predefining these relations, regardless the domain” [5]. This technology has important application like learning selectional preferences [35] or acquiring common sense knowledge [26] [14]. The domain of Open IE is introduced in section 3.1. It includes works that describe a custom developed extraction system for open domain corpora.
Although such an extraction system is a valuable contribution itself, the ultimate goal is to have *useful* data. This entails that the data has to be made available in way that provides researchers with the usability they require. Therefore, other related work in this area has the focus to provide a data resource created from large scale Open IE extractions. The data resource structures the data in a meaningful way, to target specific application areas. These works are introduced in section 3.2.

## 3.1 Open Information Extraction

The three selected works show techniques to extract open information, which differ in terms of the input corpus, the extracted data and in amounts of iterations over the corpus. The input data is different in terms of heterogeneity and size. A large scale input corpus requires highly scalable algorithms, whereas a small corpus can be analysed using advanced natural language processing techniques. Because a state-of-the-art Open IE uses natural language processing (NLP) tools [28], the corpus heterogeneity plays an important role as well [46]. This is due to the fact that most NLP tools are trained on well-formed input text and not on random chains of characters found on the web. In terms of extracted data there are also differences in the literature. On the one hand researchers use a set of manually selected patterns to identify related pairs, so called tuples. On the other hand, researchers developed systems, that extract relation independently. These systems identify so called triples, that can identify previously unknown relations and its related entities. How these systems were implemented and evaluated is described in the following subsections.

### 3.1.1 LSOE - A Lexical-Syntactic patterns based Open Extractor

In [5] the authors introduce a novel Open IE approach that extracts triples using lexical-syntactic patterns. The authors demonstrate with this work, "that an approach that performs unsupervised extraction of triples applying a few lexical-syntactic patterns in texts labelled only with POS tags, obtains results consistent with state-of-the-art". Two relatively small corpora are selected to prove their hypothesis. On the one hand a 217 sentences from documents related to the Philosophy of Language and on the other hand $2,701$ articles from the Wikicorpus [1]. The input texts were POS tagged using a third party tool and then processed by their custom developed extractor. This extractor, which is named LSOE, identifies triples of three different relation spaces by applying between three and eight patterns per relation. It is no surprise that the first corpus only led to 314 extractions and the second to $2,539$. The relatively small of output is caused by the corpus size as well as the selection of only a few patterns. However, the achieved precision of the extracted triples is considerably high with 54 percent for the domain independent Wikicorpus input and 62 percent for the domain dependent corpus. With a comparison with existing Open IE system the authors prove their initial hypothesis. Although the accuracy is comparable to the state-of-the-art, it has to be noted that the corpus is rather clean in terms of well-formed sentences and contains well-defined concepts. This makes it comparatively easy to apply lexico-syntactic patterns to the text, because the small input can be assessed beforehand and the extraction does not have to deal with malformed input, such as foreign symbols or *sentences* longer than 1 million characters. The related work introduced in the next section, is an approach that copes with such a *noisy* input corpus.

### 3.1.2 TextRunner

According to Etzioni et al. [13] the problem with most IE work is, that it is focused on a small number of relations in preselected domains, certain corpora, like newspapers or emails, and only target a small amount of relations. Thus, the researchers

[1]available at http://www.cs.upc.edu/ nlp/wikicorpus/, accessed on 10.10.2015

introduce a novel extraction paradigm "[...] that tackes an unbound number of relations, eschews domain-specific training data, and scales linearily to handle web-scale corpora [...]" [13]. To demonstrate the new domain- and relation-independent extraction approach, a system called TextRunner was implemented. TextRunner performs the domain independent extraction in two phases. First, build a model of how relations are represented in a particular language. According to the researchers an Open IE system cannot rely on a set of lexical patterns to detect relations on the web corpus. Lexical patterns are relation specific and therefore not suitable detect a previously unknown amount of relations. The authors suggest to learn a model "based on unlexicalized features such as part-of-speech tags [...] and domain-independent regular expressions [...]" The second step then extracts triples of the form *(subject, predicate, object)* from a text corpus using the model of the first phase. A test on a corpus of 120 million web pages was rather successful in terms of recall, as it returned over 500 million triples. The researchers conducted further tests in cooperation with Google and found out, that "the use of an order-of magnitude larger corpus boosts both precision and recall" [13]. During their tests the researchers observed a high heterogeneity on the web corpus. This aspect "[...] makes tools like parsers and named-entity taggers less accurate, because the corpus is different from the data used to train the tools [...]" [46]. After the extraction the researchers develop a search engine that makes the extracted triples accessible. In the work of Agichtein and Gravano [1], which introduced in the next section, the two phases are bootstrapped to an iterative process, that refines extraction patterns after each iteration.

### 3.1.3 Snowball

In this work [1] the researchers construct a system, which "introduces novel strategies for generating patterns and extracting tuples from plain-text documents" [1]. The authors carry out an experiment that focusses on the relation *located in*, between a company and a location. As input for the extraction a rather clean and small input corpus for the extraction. To be precise, the input consists of a $320,000$ news paper articles from the North American News Text Corpus, which are split into a training and a test set. In the first step of the strategy a number of seed tuples is selected. With these seed tuples text passages are identified and stored, which contain a tuple. These text passages are then used to generate patterns, that describe a *located in* relation. Using the generated patterns the system proceeds to identify new tuples. The newly extracted tuples can can then be used as seed tuples for a further iteration of the entire system. Each consequent iteration refines the set of patterns. For this purpose each pattern is assigned a confidence value. If the value is beneath a certain threshold, the corresponding pattern will be deleted and the

next iterations are performed without it. The strategy has the goal to improve patterns and extracted tuples with each iteration. The experiment is evaluated using precision and recall measures and compare them to existing information extraction systems. According to the researchers this strategy is "flexible, so that [they] capture most of the tuples that are hidden in the text in our collection, and selective, so that we do not generate invalid tuples" [1].
With such extraction systems valuable resources can be created as described in the following section.

## 3.2 Extracting structured resources from the Web

The works described in this section performs an extraction of domain independent data, and also provide useful meta-data in the form of generated attributes or provenance information. The input corpus in these works include large amounts of web documents. The following three selected works differ in terms of how the authors created:

- knowledge bases [12];
- ontologies [20];
- taxonomies [45].

According to [7] a "knowledge base is a source of information that stores facts explicitly as declarative assertions". Among others, the purposes of a knowledge base include the preservation of intellectual property or serve as repository for learning resources [42]. A special form of such a valuable information source is a ontology [19]. According to Uschold and Jasper [40] an ontology can be characterized the following way: "An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms." In contrast to a taxonomy, an ontology contains multiple relations between concepts. A taxonomy focusses on a specific relation which is often referred to as ISA or hyponymy relation. [11]. With this relation space a taxonomy can be built, by structuring these directed relations into a hierarchy [11]. In the following subsections the way how to create the above mentioned resources in automatic fashion is described.

### 3.2.1 Probase

Probase [45] is a data resource developed in 2012. It is a probabilistic taxonomy for text understanding [45]. The researches extracted concepts from a web corpus of $1.68$ billion web pages for the purpose of creating a universal taxonomy. The difference to previous approaches is the extent of the taxonomy, as it contains more than $2.7$ million concepts. This is achieved by allowing more vague concepts in form of modified noun phrases. The researchers argue, that existing taxonomies only provide well-defined concepts such as *company*, but do not include more real-world concepts such as *large company*. An additional novelty of their experiment is, that the extracted hyponymy tuples are given a certain probability, which describes the certainty that an extracted relation is true. Furthermore, this research introduced a novel framework to construct the taxonomy. Using an iterative learning algorithm, hyponyms are extracted from the web corpus. The second part of the framework is a taxonomy construction algorithm, that transforms the extracted data in a hierarchical structure. At the time of its creation, Probase was largest taxonomy regarding the amount of concepts and its probabilistic nature provides benefits to applications that require text understanding.

### 3.2.2 Biperpedia

In Gupta, et al. [20] an ontology for web search applications is introduced, which is called Biperpedia. It is an automatically created ontology, that contains $1.6$ million class-attribute pairs and $67,000$ distinct attribute names. In contrast to Probase, the goal is not the creation of a taxonomy, but a new ontology tailored for search applications.

As the experiment to create Biperpedia was performed at Google Research, the researchers are able to make use of the query stream of their search engine [2], to extract attributes for the ontology. Additionally, the researchers incorporate attributes extracted from web text as well as Freebase. Freebase is "a collaboratively created graph database for structuring human knowledge"[4], that contains "125,000,000 tuples, more than 4,000 types, and more than 7,000 properties" [4]. The extracted attributes of these data sources of structured and unstructured nature, are merged and then further enhanced. For example by attaching lists of common misspellings and synonyms. To further specify the attributes an algorithm description is provided, that can classify attributes into certain categories. With the enhanced and classified attributes a method is described to link the attributes with the most suitable classes. The experimentally measured precision of the attributes is over $0.5$

[2]User Interface available at google.com, accessed on 25.10.15

for the top $5,000$ entities. The attribute set can be regarded as valuable addition to existing ontologies, like Freebase or DBPedia, as these only contain 1 percent of the top $5,000$ attributes. DBPedia is a knowledge base that contains over $2.6$ million entities. Each entity is provided with a "human-readable definitions [...], relationships to other resources, classifications in four concept hierarchies, various facts as well as data-level links to other web data sources describing the entity" [3].

### 3.2.3 Knowledge Vault

Both above mentioned approaches use web data as input, which can be rather noisy. Therefore Murphy, et al. [12] carried out an experiment that does not only uses web data as input, but "combines extractions (obtained via analysis of text, tabular data, page structure, and human annotations) with prior knowledge derived from knowledge repositories". The result is a knowledge base, which is named Knowledge Vault: a web-scale probabilistic knowledge base. To create this knowledge base the researchers extract data in form of triples from each source, e.g. tabular data, and assign a confidence value to the extracted triples. In parallel, a graph-based approach is taken to learn the probability of each possible triple, based on triples stored in an existing knowledge base. Lastly, the extracted web triples and learned probability of each possible triple from the knowledge bases, are combined in a so called knowledge fusion. The knowledge fusion computes the probability of a triple being true, based on agreement between triples of both sources. This knowledge base is different from other ones, as it combines rather unclean web extractions with prior knowledge of existing knowledge bases. Furthermore, it is much bigger than other comparable knowledge bases according to the authors.

Unfortunately all these described resources and the exact algorithms are not publicly available. Therefore, it is an important contribution to provide such a structured resource publicly. As a first step this work sets out to create a large scale hyponymy relation database, which then can be used to create a taxonomy similar to Probase [45].

# Chapter 4

# Methodology

The methodology of this research is a set of experiments, that generates a large scale data source of hyponymy tuples. The key concept used in this experiments is a hyponymy tuple. The term hyponymy originates from the domain of linguistics and is defined by Crystal [10] as the following:

**Definition 1** *"A term used in semantics as part of the study of the sense relations which relate lexical items. Hyponymy is the relationship which obtains between specific and general lexical items, such that the former is 'included' in the latter (i.e. 'is a hyponym of' the latter)" [10].*

An example for such a hypnyom is for example the tuple $\langle apple, fruit \rangle$. These hyponymy relations consist of three components: One relation and two entities. The hyponymy relation is identified by matching lexico-syntactic and lexico-grammatic patterns with a given text. As one of the entities, the hyponym, is included in the other one, this subordinate entity is referred to as instance in this experiment. In the example the instance is the $apple$. The second entity, the hypernym, includes the first one, and is therefore the superordinate and referred to as class in this thesis. The class in the given example is $fruit$. The two entities together are referred to as tuple. How such a hyponymy database is created from a heterogeneous and large scale input corpus is depicted in figure 4.1 and will be described in the following paragraphs.

## 4.1 Extraction Workflow

Figure 4.1 describes processing steps, which are depicted as arrows, as well as the input and output data, which is depicted using cylinders. The direction of the arrows represents the sequence of execution. Each processing step is provided with

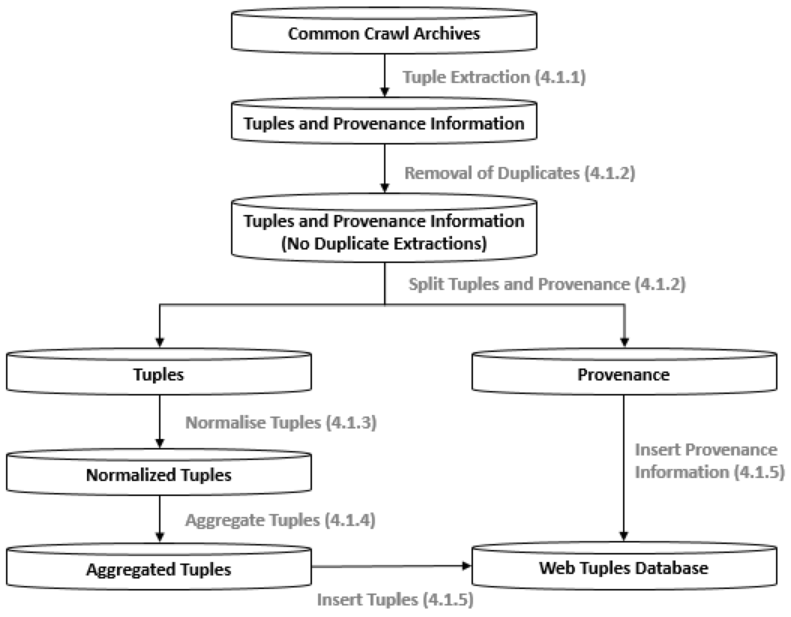
Common Crawl Archives
Tuple Extraction (4.1.1)
Tuples and Provenance Information
Removal of Duplicates (4.1.2)
Tuples and Provenance Information (No Duplicate Extractions)
Split Tuples and Provenance (4.1.2)
Tuples
Provenance
Normalise Tuples (4.1.3)
Normalized Tuples
Aggregate Tuples (4.1.4)
Aggregated Tuples
Insert Provenance Information (4.1.5)
Web Tuples Database
Insert Tuples (4.1.5)

Figure 4.1: Workflow for the WTDB Creation

a name and a reference to the section, in which the processing step is described.
The initial input data consists of archives from the Common Crawl. The Common Crawl is "an open repository of web crawl data hat is universally accessible and analysable" [16]. These archives contain the text of web pages and are used to extract tuples in the step named *Tuple Extraction*. By using patterns that can detect hyponymy relations, tuples are extracted from the archives. The result of the first processing step are the tuples along with information about their extraction and their provenance information. In the following processing step *Removal of Duplicates* the provenance information is then used remove tuples, that were extracted from the same sentence under the same pay-level-domain (PLD). This processing step reduces the data volume, without losing any unique tuples and reduces the impact of spam. After the duplicate removal the provenance information is split from the tuples in step *Removal of Duplicates* as it is not further processed and would slow down processing. Due to the heterogeneous nature of web data the tuple extraction as well contains heterogeneous representations of entities. Heterogeneous in this context means, that the same entity can be written in different ways. For the single tuples to be aggregated however, the string representation of the tuples has to be equal. Thus a normalization of the extracted entities is performed in step *Tuple*

*Normalization*. The goal is to reduce the amount different written forms of entities. The process *Aggregate Tuples* then identifies tuples with equal string values of classes and instances and merges them. During the merge the frequency of a tuple, the amount of distinct patterns and amount of distinct pay level domains for each tuple is identified and stored. The output of this process are unique tuples. These are inserted in a database management system in step *Insert Tuples* to enable the access for researchers. Due to the size of the data it is split across over thousands of tables to enable decent query execution.
During each of these steps the challenge is to find a trade-off between the runtime and the complexity of the used algorithms, given the size of the dataset and the restrictions of the technical infrastructure. To describe how this challenge was overcome each processing step is documented in detail in the following subsections.

### 4.1.1 Tuple Extraction

This subsection describes how tuples are identified and extracted from a large corpus, namely the Common Crawl archives [1].
The Common Crawl Foundation crawls the web multiple times a year, stores the crawled data and makes these archives publicly available on the web [2]. The data is stored at Amazon's S3. S3 is the abbreviation for Simple Storage Service, which is a commodity-priced storage utility [31]. The S3 can be classified as Storage-as-a-Service (STAAS), because it enables users to store and access data not on their own, but on Amazon's infrastructure [44]. It has to be noted, that these archives do not contain every piece of information available on the web. The archives are built from data available on the web using a so called web crawler. "A web crawler is a program that downloads web pages" [8]. The crawler developed by the Common Crawl Foundation respects crawling conventions, stored in *robot.txt* for example and abides *nofollow* instructions of embedded hyper-links [32]. These archives are available in three formats, which are briefly described in the following. "The WARC format is the raw data from the crawl, providing a direct mapping to the crawl process. Not only does the format store the HTTP response from the websites it contacts [...], it also stores information about how that information was requested [...] and meta data on the crawl process itself [...]." [17]. The data stored in the WARC archives is split and then stored in WET and WAT archives. The WAT archives contain meta data that includes the HTTP headers returned and the links listed on a page [17]. WET archives on the other hand contain the extracted

[1]Archive links available at https://aws-publicdatasets.s3.amazonaws.com/common-crawl/crawl-data/CC-MAIN-2015-18/wet.paths.gz, accessed on 25.10.15

[2]Overview of crawl data: https://commoncrawl.org/the-data/get-started/, accessed on 25.10.15

plain-text of a web page [17] and meta data including the URL of the web page. For this experiment the WET archives are selected, because only the provenance information as well as the extracted plain-text of a website are required. The latest crawl of the Common Crawl was selected, which was performed in April 2015. It is selected because it has the most up-to-date information compared to earlier crawls. Furthermore it is assumed that the amount of crawled web pages is growing from crawl to crawl, because of the exponential growth of information on the web. This crawl contains more than 2.11 billion web pages and has a size of 168 terabytes (TB). The textual data of these archives is analysed using a set of lexico-syntactic patterns. How these patterns are selected and used in order to achieve a high recall, is described in the following sections.

#### 4.1.1.1 Pattern Collection

To acquire patterns that indicate a hyponymy relation between two noun phrases, existing literature on hyponymy extraction is analysed. Each literature work and their respective lexico-syntactic patterns are described in the following paragraphs. Based on a rough evaluation, it was decided to adapt a pattern or not. Such an pre-evaluation is necessary, because some of the patterns have been extracted automatically in Open IE experiments. Thus, these patterns have not yet been applied for hyponymy extractions. Moreover, some of the automatically generated patterns occur rather rarely or only in specific domains. With these aspects in mind and the guidelines of high recall and fast extraction, it is acceptable to remove these *exotic* patterns. Since they don't frequently occur, the recall will not be harmed significantly. The runtime of the extraction benefits from each excluded pattern equally, so removing low recall patterns is a performance increase.
Hearst [21] identifies "a set of lexico-syntactic patterns that are easily recognizable, that occur frequently across text genre boundaries, and that indisputably indicate the lexical relation of interest" [21]. These patterns are used to interpret unrestricted, domain-independent text and are therefore all deemed suitable for this experiment.
Ponzetto and Strube [33] set out to induce a taxonomy using lexico-syntactic matching and the categorization system of Wikipedia. The researchers present methods to "automatically assigning isa and notisa labels to the relations between categories" [33]. For this task the researchers use two sets of patterns. On the one hand patterns that indicate a *notisa* relation and on the other hand a set of patterns that indicate an *isa* relation. A *isa* relation is identified using hyponymy patterns, whereas a *notisa* relation is found using meronymy patterns. The latter set of *notisa* tuples is used, to prevent a wrong labelling of *isa* relations. The refined *isa* rela-

tions can then be used to automatically create a taxonomy. The set of *isa* patterns is selected for this experiment.
A study by Orna-Montesinos [30] set the goal to find "lexico-grammatical patterns which signal the hyponymy relations of the term building." As corpus a set of specialized textbooks in the field of construction engineering and construction is selected. The study not only reveals new patterns that indicate a hyponymy relation, but also validated the existence of manually selected patterns, as for example the ones of Hearst [21]. Overall, this study provides an extensive list of lexico-grammatical patterns indicating a hyponymy relation. Although, these patterns were acquired from a domain specific corpus, they do not contain any domain-specific vocabulary and are therefore mostly rated as suitable patterns.
Furthermore, the research of Klaussner and Zhekova [25] provides useful patterns. Their research describes "how ontologies can be built automatically from definitions obtained by searching Wikipedia for lexico-syntactic patterns based on the hyponym relation" [25]. For this purpose the researchers provide a set of lexico syntactic patterns that have proven to be successful. The goal of their research is to extract definitions from domain-independent, unrestricted text. The used patterns are selected for this this project.
Further suitable patterns are identified in the study of Ando et al. [2], in which a method is introduced, that automatically extracts hyponyms from Japanese newspapers. Of course it seems odd to apply patterns used in Japanese newspapers, to identify hyponyms in the English language. However, the English translations of these are capable of finding hyponyms in English texts. Since an acceptable recall and accuracy can be expected from these patterns, they selected for this research, as well.
The above identified patterns are by no means unique. For example, the patterns identified by Hearst [21] can also be found in the research of Ponzetto and Strube [33], Orna-Montesinos [30], Klaussner and Zhekova [25] and Ando et al. [2]. Thus these overlaps are removed, because it does not make sense to match the same pattern multiple times. The list of all selected patterns is provided in a table in Appendix B. The patterns there are described in a custom notation, which is used to highlight the position of the instance, the keywords of the pattern and the position of the class. An instance position is indicated by the abbreviation $NP_I$, which is short for noun phrase of the instance. The keywords of the hyponymy pattern are written in lower case letters and the position of class term is indicated with the abbreviation $NP_C$. Each pattern is assigned a unique ID and a source, which describes where each pattern can be found in literature. This pre-selection of patterns is now translated into code and applied to a test corpus, to determine their usability for a large scale hyponymy extraction. How these patterns are translated into code is described in the following section.

#### 4.1.1.2 Regular Expressions

Before a first test with identified patterns can be executed, these have to translated into a language suitable to parse text. Regular expressions (regex) form such a language, that is used to parse and manipulate text [38]. Usually regex are used to carry out complex find-and-replace operations [38]. As this language suitable to parse the provided text data and as it is integrated into the programming language of the framework, the choice is made to detect patterns using regex. For this regex translation the patterns are categorized into specific pattern types. These types are different in terms of the position of pattern keywords. The position of keywords then entails the position of the class term and the instance term. Four types of patterns can be distinguished with the position of keywords:

- Compact Patterns
- Split Patterns
- Isolated Patterns
- Apposition Patterns

How these different pattern types are defined and translated is described in the following paragraphs.

**Example Compact Pattern:** $NP_i$ or other $NP_c$

The keywords of the *compact* pattern are positioned next to each other, without any words in between. An example for such a pattern is provided above, in which the keywords are *or other*. These pattern types are translated into regex using the following schema. As we search for classes and instances indicators for such entities are required. Such an indicator can either be a unicode letter or a number. Numbers are allowed in order to detect entities like *Windows 7* or *76ers*. However certain symbols other than letters and numbers can occur in between words. The selected symbols include all types of quotation marks, apostrophes and the trademark symbols. The more symbols are allowed the more recall is expected. However, the performance of the regex matching will be reduced, with an increasing number of symbols. Therefore only symbols were selected per hand that are expected to occur frequently in English sentences. Thus, after the first indicator of an entity a list of optional symbols is allowed. Additionally, the regex for compact patterns allows an optional comma. Even if some compact patterns have no place for an optional comma it is still introduced for two reasons. First, the patterns are easier maintainable if they follow a common structure. Second, it can be assumed that a significant proportion of sentences on the web are not written in proper English grammar in terms of comma usage. Therefore, it was decided to include an optional comma to increase the recall without a significant performance decrease. The optional comma is followed by a space and the keywords of a pattern. If the pattern contains multiple keywords these are separated by a space. The keywords are then followed by a space. As previously stated, an entity in a sentence might be surrounded by quotation marks. Therefore, an optional symbols are allowed in front of the indicator for the latter entity, which again can start with a letter or a number. A list of regular expressions for compact patterns is provided in Appendix C.

**Example Split Pattern:** Such $NP_c$ as $NP_i$

Second, the so called *split* pattern has the characteristic that either a class term or an instance term is between the keywords of the pattern. An example for such a pattern is provided above, in which the key words *such* and *as* are split by the words of the class. With one entity being surrounded by keywords, a simple indicator that consists of one single character is not suitable. Thus, an entire place holder for an entity has to be integrated into the regex. The place holder for an entity contains at least one word and a maximum four words. Single words are separated by spaces. A word however does not only contain letters. Also hyphenated words, such as *well-known* are detected, as optional hyphens are integrated into the regex describing a single word. As described for compact patterns, also split ones have to take into account that words may start or end with optional symbols. The same optional symbols used in the compact pattern are applied for these split patterns. A special case for split patterns has to made in the case of patterns that start with an adjective in the superlative. If an adjective contains more than two syllables the superlative of the adjective usually contains two words: On the one hand *most* and on the other hand the adjective itself. Therefore, split patterns that make use of the adjective in superlative form, have to contain at least two words and up to five words, as one of these words is reserved for the adjective after *most*. A list of regular expressions for split patterns is provided in Appendix D

**Example Isolated Pattern:** $NP_c$, $NP_i$ for example

The third type of patterns is named *isolated* pattern in this thesis. This name is chosen, because neither do the keywords surround an entity, nor do the entities surround a key word. One could say the entities and keywords are *isolated* from one another. The keywords of these patterns are positioned behind the entities for classes and instances. An example for such a pattern is provided above. For the isolated patterns, the already developed place-holder from split patterns is used for both, the class and the instance noun phrase. After an optional comma behind the last entity place holder, the keywords are specified.

**Example Apposition Pattern:** $NP_c, NP_i,$

Lastly, the apposition pattern is described. In this case it is not keywords that indicate a hyponymy relation, but punctuations. In this cases of an apposition the instance term is followed by a class term, that is surrounded by punctuations. An example is provided above, in which commas indicate the apposition pattern. Therefore the regex for apposition patterns starts like a compact pattern. An indicator for an entity is required in front of the punctuations of the apposition. As before, this indicator can either be a number or a letter followed by optional symbols. Next is the punctuation symbol of the apposition followed by a space. Equal to split patterns a place-holder for an entity is required between the punctuations, as described in the paragraph above. The place-holder is then followed by the punctuation symbol of the apposition, which also signals the end of the regex. There is no list of apposition patterns provided, because these were excluded from the experiment. The symbols used for apposition patterns simply occur too frequently in the corpus, which would prevent an efficient filtering of input data. The reasons and used methods for filtering are described in section 4.1.1.3.

The regular expressions for the place-holder are further optimized to ensure a robust matching process and reduce its duration. The regex for the place-holder has to contain quantifiers to represent multiple characters. The standard quantifiers of regular expressions are greedy, which means they try to match as many times as possible [38]. This is an adverse configuration for this experiment, because the regex contain key words that occur after the place-holder. As a consequence, the matcher will match as many letters as possible. In this case it matches the entire input line, until a not allowed character is found. After that the matcher will backtrack character by character and try to match the key words. This massive runtime increase can not be justified and is prevented by the usage of capturing groups and possessive quantifiers. These ensure, that if a place-holder is matched, the key words are tried to be matched immediately afterwards. This way a pattern can be found, with only one pass over the characters of a line. In case of a matched pattern the entire line is processed again using more advanced NLP techniques, as described in the following section.

#### 4.1.1.3 Splitting and Filtering Input Data

Given the noisy nature of the Common Crawl data, it is not feasible to perform a test run of these regex. As the corpus is read line by line, the length of line has an impact on the performance of a regex. Lines longer than $100,000$ characters are rather frequent in the corpus and therefore have to be split before a regex can be matched. The best way for this experiment is to use a sentence splitter to split each line into sentences and then analyse the single sentences using regex. This way tuples can not be destroyed, because the identified patterns are within the confines of one sentence. For this purpose a regex is used to split the line sentences. The first indicator of the end of a sentence are the following punctuations: The full-stop ".", the question mark "?" or the exclamation mark "!". If one of these punctuations is followed by a space and a capital letter, then the line is split. Similarly to the regex of the patterns, sentences can start or end with optional symbols such as the quotation mark. These are considered for the sentence split regex, as well. Additionally, the sentence splitter is implemented in a way that it does not split common abbreviations, that are using a full-stop. For example, the name *Marti A. Hearst* would be split in two sentences after the *A.*, if common abbreviations are disregarded. For this purpose a list of common abbreviations from the European Parliament Proceedings Parallel Corpus (Europarl) [3] is used and extended manually. After the first test runs of the extraction, it became evident that a few abbreviations, such as *fig.*, are used rather frequently in the input corpus. If an abbreviation is observed more than once in a test extraction, it is added to the list, which is provided in Appendix F.
With the patterns being in the English language, a website filter can be implemented, that filters out documents from websites, which are written in a foreign language. Such a filter would decrease the amount of input data significantly and therefore speed up the extraction. But due to the nature of the corpus, such a filter can not be implemented without risking the loss of valuable tuples. A single web page can contain multiple languages. For example users comment on an English news article in their native language. Thus, the language detection tool has to be applied to each section of a web document. Even if it is possible to detect each section of a web document and apply a language detection algorithm, it would drastically increase the runtime. With many colloquial terms and *not* natural language text passages on the web, the language detection also would become susceptible to errors. Such a runtime increase, with questionable results cannot be justified. Hence, web pages are not filtered according to language criteria.
Even after the introduced sentence split, there still remain text passages, that are too long for a robust regex execution. Thus, an additional *text chunker* is implemented,

[3] Available at http://www.statmt.org/europarl/, accessed on 25.10.2015

that splits a text passage after a fixed amount of characters. The more characters are allowed in a text passage, the less splits are performed. With less splits the probability is decreased, that a valid tuple is split in half and therefore would become invalid. Consequently the amount of allowed characters should be as large as possible. The character threshold is determined empirically. The threshold is increased with each test run until, until one of the regex fails to match the line. The resulting maximum length was set to 400 characters. Additionally a minimum sentence length of ten characters is introduced, to reduce processing time. This is due to that fact, that even the shortest pattern, requires at least 4 characters and two spaces. Therefore, one can safely assume, that the remaining four characters will not provide too much useful information.
With the lines chunked in processable chunks the first test runs still show poor performance and are therefore not suitable to be run on the entire corpus. Especially the regular expressions containing a place-holder for an entity performed poorly. The solution to this problem is a pre-check method, which determines if a line is worthy to analyse with the rather complex regex of the patterns. This method takes a line chunk as input and checks, if one of the keywords, e.g. *such as*, of the patterns is present in a line. The method turned out to be successful on web data, because the web does not only contain well formed sentences. Timetables of bus stations, lists of personnel or price lists are examples for lines, that contain thousands of characters without any useful information for this experiment. However, the pre-check method has a significant downfall: apposition patterns can not be pre-checked reliably, because these punctuations occur not only in sentences, but also in lists and timetables. To make it worse, these apposition patterns contain a place-holder for an entity, which entails an increased matching time. Consequently the decision is made to exclude all apposition patterns from this experiment, to ensure a robust tuple extraction.

The extraction of single entities requires more complex algorithms and therefore should only be executed if the tuple provides useful or novel information. To determine if a tuple contains useful information a simple heuristic is used. If the words directly in front of or behind the matched pattern are pronouns or interrogative words, the match is ignored and the next sentence is read. A pronoun is a substitute for a noun and usually can be resolved using its context. But such a resolver is computationally to costly for a fast extraction. Since unresolved pronouns do not provide useful information for this experiment, these are filtered out. The same applies to interrogative words, such as *what*. Furthermore, if a sentence is found more than once under the same pay level domain, it will not be analysed again, as we already have stored the information. This will reduce the influence of automatically generated content on the frequency measurements of tuples on archive level.

At this stage the regular expressions are defined, the input is split in reasonable chunks and a pre-check method is implemented. Hence, a sample extraction is performed on a subset of corpus to measure the recall and the precision of the constructed regex. A random segment of the corpus is selected, that has approximately the size of 1 percent of the entire corpus. If one of the regex matches a sentence, the matched line is stored and analysed for valid hyponymy relations. The guidelines used for this evaluation are taken from Ponzetto (2011) [33] and are also provided in Appendix B. For the precision evaluation 100 matches per pattern were analysed, or all matches of a pattern if it had less than 100. The results of this evaluation are provided in Appendix B and are discussed in the following. Each pattern had at least twelve correct matches, which indicates the validity of the identified patterns and their regex translation. Unfortunately three patterns that have an precision below 20 percent, have an massively increased recall, compared to other ones. These are the patterns $NP_c$ *like* $NP_i$ (p9), $NP_i$ $NP_c$ *types* (p35) and $NP_i$ *as* $NP_c$ (p40). Such a high recall is not a surprise, if one considers the frequent usage of the keywords in natural language texts. Using the the recall of the text crawl and the precision of 100 analysed tuples an estimation is made, how many false positives these patterns will return. The pattern with the ID *p35* matches more than 3000 false positives per archive, the pattern with the ID *p9* more than 16000 and the pattern with the ID *p40* more than 36000. Although a guideline of this experiment is a high recall, it is decided to remove these patterns from the experiment. This is due to the reason, that further processing steps get computationally more expensive and therefore either require more processing time or better technical resources. With both, time and resources strictly limited for this experiment, it is the safest decision to disregard these patterns. The isolated patterns had to be excluded due to bad matching performance. These are patterns, in which the instance and the class terms are not separated by a pattern keyword, namely "$NP_C$, $NP_I$ for example" (p32), "$NP_C$, $NP_I$ for instance" (p41) and "$NP_C$, $NP_I$ and the like" (p44). The explanation for this behaviour is simple. Since the first parts of the regex are place-holders for noun phrases, these parts basically match any words in a sentence. Then the matcher tries to match the keywords, which are more strict, as they only allow a fixed set of characters in a exact order. The result is, that the matcher spends a lot of time on matching the place-holders, and then fails most of the time, when trying to match the key words. To ensure the robustness of the system, the safest decision is to exclude these three patterns from the extraction system.
If one of the patterns successfully matches a input text, the next step is to extract single entities from it. This processing step is described in the following section.

#### 4.1.1.4 Noun Phrase Extraction

At this stage, the extraction system is able to find sentences or line chunks, that contain a potential hyponymy relations. However, it is not the goal to extract sentences, but single tuples. So the next step is to extract single noun phrases in front of, after or inside the matched pattern. A noun-phrase can be defined as follows:

**Definition 2** *"The constructions into which nouns most commonly enter, and of which they are the head word, are generally called noun phrases (NP) or nominal groups. The structure of a noun phrase consists minimally of the noun (or noun substitute, such as a pronoun); the constructions preceding and following the noun are often described under the headings of premodification and postmodification." [10]*

So the next challenge is the extraction of noun phrases. Each noun phrase must contain a noun or a substitute. With noun substitutes excluded from this extraction, it is the first step to find words, which are tagged as nouns. Approaches using dictionaries of nouns are not suitable for this experiment, because the input corpus is too large. The single words of the matched sentences would have to be compared to the words in a dictionary. Such an approach does not scale as required. In previous research in the area of Open IE the input texts are part-of-speech (POS) tagged using third party tools [5] [33] [14]. As these approaches have been successfully tested on web documents [5] before, this research employs the same approach. Therefore POS tagging is used, which is:

**Definition 3** *"POS tagging identifies the lexical category of each word in a sentence on the basis of its context. An accurate definition of POS tagging (POST) is a mapping from a sequence of words to a sequence of lexical categories." [34]*

With such a mapping of words to lexical categories, nouns can be identified and therefore a noun-phrase with its modifications can be extracted. This basically entails a selection of allowed POS tags for the modifiers and the head noun. The list of possible POS tags is dependent on the used POS-tagger. In this research the POS tags of the PENN Tree Bank [4] are used. Unfortunately, the POS tags are not completely suited for this purpose. The problem is, that some tags can either refer to a verb or a participle. While the participle form can be a modification for a noun phrase, the verb cannot. With many ambiguous tags the true confines of a noun phrase are difficult to detect. As a result, an unrestricted detection of noun phrases

[4] List of POS Tags available at https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html, accessed on 25.10.2015

| Name | Tag | Example |
|---|---|---|
| Noun singular or mass | NN | apple |
| Noun plural | NNS | apples |
| Proper noun singular | NNP | Spain |
| Proper noun plural | NNPS | Spaniards |

Table 4.1: Allowed Tags for the Head Noun

only using POS tags, will result in invalid and rather long noun phrases. Thus a maximum allowed length is empirically determined and set to four words per noun phrase. The allowed head noun tags include every form of nouns or proper nouns and are depicted along with descriptions and examples in table 4.1.

However, an additional constraint is implemented, which ensures that a noun in genitive form is not categorized as head noun. These nouns are then manually labelled as adjective, to ensure these are included in the noun phrase, but not as its head. It is important to note, that the head noun is not always the first noun in a noun phrase. This is the case for chains of nouns, in which the last noun is the head noun. Take for example the noun phrases *bus driver* or *bus driver union*. In both cases not the *bus* is the head, but the last nouns in each chain. The subsequent nouns are stored in the post-modifier, although these are the actual head.
As no exhaustive lists of allowed POS tags for pre-modifiers could be found in literature, the tags were manually selected. As the goal of this experiment is information extraction, only POS tags are included the have the potential to enrich or specialize the information about the noun phrase. Therefore, any kind of pronoun, determiner or quantifier is excluded from the list of allowed POS tags for the pre-modifier. Forms of participle verbs hold valuable information. For example the present participle in *the flying dutchman* or the past participle as in *the recovered artefact*. Unfortunately the used set of POS tags does not allow the differentiation between verbs in the gerund form and the present participle form. To avoid the integration of verbs into the pre-modifier, the present participle form is removed from list of allowed POS tags for the pre-modifier. Additionally, adjectives are included in the pre-modifier as they provide more specificity to the noun phrase. The resulting list of pre-modifiers is provided in 4.2.

The tags of the post-modifier are listed in table 4.3. It includes all allowed tags of the pre-modifier and the head noun. The first reason to include all kinds of nouns is the presence of noun chains, for example *bus driver*. Additionally, a noun phrase can be specified by a preposition in combination with a noun, as in *American declaration of independence*. Consequently prepositions are categorized

| Name | Tag | Example |
|---|---|---|
| Adjective | JJ | new |
| Adjective comparative | JJR | newer |
| Adjective superlative | JJS | the newest |
| Verb past participle | VBN | elected |

Table 4.2: Allowed Tags for the Pre-Modifier

as valid post-modifier tags, although these do not hold any specific information. Therefore prepositions can be regarded as indicator for more specific information in the subsequent words of the sentence. The same applies to determiners. For example, *member of the senate* provides more information, than *member*. Furthermore cardinal numbers as in *iPhone 6* are added to the list. The present participle form of verbs is also integrated in the post-modifier list, if it is followed by a noun. The assumption is that the present participle is a specification of the following noun.
After the post-modifiers have been detected they have to be corrected in particular cases. This is the case if the post-modifier does not end with one of the noun types or a cardinal number. The remaining possible POS tags would either hold no information about the noun-phrase, like the preposition *of* in *best basketball player of*. Thus, each word of the post-modifier is removed, until the last noun or cardinal number is found.

After the modifiers are identified, it may occur that the maximum length of four words per noun-phrase is violated. In these cases the decision is made to remove the pre-modifier. By no means does that mean, that the pre-modifier carries less information than the post-modifier. However, in case of nouns chains, the head noun of the noun phrase is positioned in the post-modifier. As the head noun has the highest priority of all NP constituents, this decision is the most reasonable. With the tags defined, the next step is to detect the noun phrases in a matched sentence, which form the hyponymy relation.

In order to determine, which noun phrases of a sentence are in a hyponymy relation to each other requires a syntactic analysis of the sentences. Given the magnitude of input data and the length of lines, the algorithm has to scale. Consequently a full syntactic analysis of the sentence is not feasible. Furthermore, the precision of such an analysis is questionable, if it is performed on unclean web data which does not only contain well-formed sentences. Therefore a rather simple approach is taken, which extracts the *closest* noun phrases to the pattern. The concept

| Name | Tag | Example |
|---|---|---|
| Adjective | JJ | new |
| Adjective, comparative | JJR | newer |
| Adjective, superlative | JJS | newest |
| Verb, past participle | VBN | elected |
| Verb, gerund or present participle | VBG | working |
| Noun, singular or mass | NN | apple |
| Noun, plural | NNS | apples |
| Proper noun, singular | NNP | Spain |
| Proper noun, plural | NNPS | Spaniards |
| Preposition or subordinating conjunction | IN | from |
| Cardinal number | CD | 1987 |
| Determiner | DT | the |

Table 4.3: Allowed Tags for Post-Modifier

of the closest noun phrase is similar to the one described in [45]. This approach is based on the assumption, that first noun phrase in front of and the first one behind the pattern are in a hyponymy relation. False extractions are therefore possible, but due to the magnitude of data, these false extractions are expected to have a rather low frequency and therefore can be filtered out at later stages.
Furthermore, one single pattern can identify multiple hyponymy tuples in case of co-ordination, which is:

**Definition 4** *A co-ordination is a term in grammatical analysis to refer to the process or result of linking linguistic units which are usually of equivalent syntactic status, e.g. a series of clauses, or phrases, or words. [10]*

The coordinating conjunctions used to detect multiple noun phrase are *and*, *or*, *&*, as well as commas. Although some proper nouns may contain these words, e.g. *Smith & Wesson*, there are no efforts taken to resolve these. It is not reasonable to use dictionaries of proper nouns or even a syntactic analysis, because the performance would suffer for a diminutive amount of proper nouns. The result of ignoring these proper nouns is demonstrated in the following. If the company *Smith & Wesson* is detected as instance, it will be extracted as two instances, namely *Smith* and *Wesson*. Co-ordinations affect the class terms as well as the instance terms. Thus, the decision is made to extract multiple noun phrases on each side of the pattern, if a coordinating conjunction is found on that side.
There are two different sets of algorithms used to detect the noun phrases. One set

for the extraction *in front* of the pattern and another for the noun phrases located *after* the pattern. Therefore, the sentence is split in two parts, which are used as input for these algorithms. For compact patterns the split is performed at the position of the pattern. In case of split patterns, the split is performed at the position of the single keyword, which separates classes and instances.

In the next paragraph the algorithm to extract noun phrases *after* the pattern is described.

**Algorithm 1** Algorithm to detect Noun Phrases after Patterns

```
procedure GETNPAFTER(List<TaggedWords> TWS, Integer offset, Boolean strict)
    $position \leftarrow offset$
    while $position < |TWS|$ do
        $TW \leftarrow TWS[position]$
        if $TW_{Tag} \notin HeadNounTags$ AND
          $TW_{Tag} \notin PreModifierTags$ AND
          $strict = true$ then
            break
        end if
        if $TW_{Tag} \in HeadNounTags$ then
            $HeadNoun \leftarrow TW$
            $getPreAfter(TWS, position - 1)$
            $status \leftarrow getPostAfter(TWS, position + 1)$
            $Persist(PreModifier, HeadNoun, PostModifier)$
            $Clear(PreModifier, HeadNoun, PostModifier)$
            if $status > 0$ then
                $getNPAfter(TWS, status, true)$
            end if
            break
        end if
        $position \leftarrow position + 1$
    end while
end procedure
```

Each algorithm consists of three functions. The algorithm to detect noun phrases *after* the pattern is initialized with the procedure *getNPAfter* and is depicted as pseudo-code in the algorithm 1. It detects the closest noun by comparing the tags of the input line with the allowed tags of the head noun. The first input parameter is a set of tagged words. A word in this case is a string, that is surrounded by spaces. The second input parameter, the offset, describes which word tag is checked next for valid head noun tags. The last input is a boolean value, that indicates if a noun phrase has already been detected. If this is the case, the consequent noun phrase detections have more strict termination criteria. When the procedure is called, each word is checked in its original order. If a POS tag is one of allowed noun tags, this noun is set as head noun. As soon as a allowed head noun tag is found the procedure passes the position of the head noun to two further functions. One function is used to detect the pre-modifier and one to detect the post-modifier. After a noun phrase with its modifications has been detected it is persisted. The variables used to temporarily store the modifiers and head noun are then cleared, which means all their associated values are removed. In the case of co-ordinations, which are found during the post-modifier detection, the procedure calls itself, with the parameter *strict* set to true. If this parameter is true, the noun phrase detection is more strict and terminates if the next word has neither a head noun tag nor a pre-modifier tag. The detection of the pre-modifier is described in the following.

**Algorithm 2** Algorithm to detect Pre-Modifiers after Patterns

```
function GETPREAFTER(List<TaggedWords> TWS, Integer offset)
    $position \leftarrow offset$
    while $position \geq 0$ do
        $TW \leftarrow TWS[position]$
        if $TW_{Tag} \in PreModifierTags$ then
            $PreModifier \overset{+}{\leftarrow} TW$
        else
            return
        end if
        if $|PreModifier| = 3$ then
            return
        end if
        $position \leftarrow position - 1$
    end while
    return
end function
```

The pre-modifier identification for noun phrases after the pattern is depicted in algorithm 2. This function is takes as input a set of tagged words and an integer value called offset. This offset points to the word, which is positioned in front of the head noun. Then in reverse order the words are checked for valid pre-modifier tags. If a such a tagged word is found, it is added to the pre-modifier, otherwise the function terminates. As soon as the pre-modifier has reached a length of three the function terminates, because the maximal noun phrase length has been reached. The detection of the post-modifier described below is more complex, because it also handles the detection of co-ordinations.

**Algorithm 3** Algorithm to detect Post-Modifiers after Patterns

```
function GETPOSTAFTER(List<TaggedWords> TWS, Integer offset)
    $position \leftarrow offset$
    while $position \leq |TWS|$ do
        $TW \leftarrow TWS[position]$
        if $TW_{Tag} \in PostModifierTags$ then
            $PostModifier \overset{+}{\leftarrow} TW$
        else if $TW_{Tag} \in Coordinations$ then
            return $position + 1$
        end if
        if $|PostModifier| = 3$ then
            return $-1$
        end if
        $position \leftarrow position + 1$
    end while
    return $-1$
end function
```

The post-modifier detection takes as input a set of POS tagged words and an integer value. The integer value is named offset and points to the word after the detected head noun. The tagged words are checked in original order to identify allowed post-modifier POS tags. If such a tag is found, the tagged word is added to the post-modifier. There are three termination criteria for this function. The execution is terminated, as soon as the post-modifier has reached a length of three. Because in combination with the head noun the maximum noun-phrase of four is reached. Furthermore, the post-modifier detection terminates as soon as a tagged word is found, that does not have a valid post-modifier tag. For both previously mentioned termination criteria, the algorithm returns an negative integer value, which indicates that no co-ordinations were found. The detection of a co-ordination is the last termination criteria and in this case an integer value is returned, that points to the next word after the coordinating conjunction. As already described, if a coordination has been found, the procedure depicted in 1 is started with more strict termination criteria. As soon as no further noun phrases

are found the entire algorithm to extract noun phrases after a pattern is finished. The extraction system then proceeds to extract noun phrases in front of the pattern. This algorithm is different, because the words are traversed in reverse order.
The decision to traverse words in reverse order for noun phrase detection *before* the pattern is made for performance reasons. It is not reasonable to start analysing tagged words starting with the first word of the sentence. The goal is to detect the closest noun phrase before the pattern. Thus one can assume, that the noun phrase is closer to the pattern, than to the start of the sentence. So if the algorithm starts at the position of the pattern and then moves gradually backwards towards to beginning of the sentence, less POS tag comparisons have to be performed.

**Algorithm 4** Algorithm to detect Noun Phrases in front of Patterns

```
procedure GETNPBEFORE(List<TaggedWords> TWS, Integer offset, Boolean strict)
    $position \leftarrow offset$
    if $strict = false$ then
        $TWS \leftarrow Collections.reverse(TWS)$
    end if
    while $position < |TWS|$ do
        $TW \leftarrow TWS[position]$
        if $TW_{Tag} \notin PostModifierTags$ AND $strict = true$ then
            break
        end if
        if $TW_{Tag} \in HeadNounTags$ then
            $HeadNoun \leftarrow TW$
            $status = getPreBefore(TWS, position + 1)$
            $getPostBefore(TWS, position - 1)$
            $Persist(PreModifier, HeadNoun, PostModifier)$
            $Clear(PreModifier, HeadNoun, PostModifier)$
            if $status > 0$ then
                $getNPBefore(TWS, status, true)$
            end if
            break
        end if
        $position \leftarrow position + 1$
    end while
end procedure
```

The algorithm for noun phrase detection *before* the pattern starts with the procedure named *getNPBefore*, which is depicted in 4. Among others this procedure takes as input a set of tagged words, which are located in front of the pattern. Additionally an offset is required as input, which points to the word that has to be analysed first. The last required parameter is a boolean value. This boolean value is used to modify the termination criteria of the procedure, in case of co-ordinations. When the procedure is initialized the first time for a line the *strict* parameter is set to false and the set of tagged words is reversed. The procedure then traverses through the reversed tagged words and checks these for valid head noun tags. If a such a tag is found, the tagged word is set as head noun and functions are called to detect pre- and post-modifiers. After the modifiers have been set, the entire noun phrase is persisted and the variables are reset. If traversed in reverse order, the co-ordination detection takes place in the pre-modifier, to determine if there are possible noun phrases in front of the closest one. If such a co-ordination is found, the procedure starts itself again, using the position of the co-ordination as new starting point.

**Algorithm 5** Algorithm to detect Pre-Modifiers in front of Patterns

```
function GETPREBEFORE(List<TaggedWords> TWS, Integer offset)
    $position \leftarrow offset$
    while $position < |TWS|$ do
        $TW \leftarrow TWS[position]$
        if $TW_{Tag} \in PreModifierTags$ then
            $PreModifier \overset{+}{\leftarrow} TW$
        else if $TW_{Tag} \in Coordinations$ then
            return $position + 1$
        else
            return $-1$
        end if
        if $|PreModifier| = 3$ then
            return $-1$
        end if
        $position \leftarrow position + 1$
    end while
    return
end function
```

The pre-modifier detection is depicted in algorithm 5 and takes as input a list of tagged words and a integer value, which points to the next word to be analysed.

The function checks the tagged words for valid pre-modifier tags and adds them to the pre-modifier. If a word does not qualify as pre-modifier, but as coordinating conjunction, the function returns an offset pointing to the next word behind the conjunction. Otherwise the function returns a negative value. On the one hand this is the case, when neither a valid pre-modifier tag nor a coordinating conjunction is detected. On the other hand the function terminates, when the maximum pre-modifier length of three words is reached. Regardless of the return value, after the pre-modifier detection has finished, the post-modifier for the current noun phrase is detected.

**Algorithm 6** Algorithm to detect Post-Modifiers in front of Patterns

```
function GETPOSTBEFORE(List<TaggedWords> TWS, Integer offset)
    $position \leftarrow offset$
    while $position \geq 0$ do
        $TW \leftarrow TWS[position]$
        if $TW_{Tag} \in PostModifierTags$ then
            $PostModifier \overset{+}{\leftarrow} TW$
        else
            return
        end if
        if $|PostModifier| = 3$ then
            return
        end if
        $position \leftarrow position - 1$
    end while
    return
end function
```

The algorithm for post-modifier detection for noun phrases located in front of the pattern is summarized as pseudo-code in Algorithm 6. The function takes as input a list of tagged words, as well as an integer value named *offset*, which points to the first word to be analysed. Starting with the tagged word identified by the *offset*, the words are checked in reverse order for valid post-modifier tags. If a valid valid post-modifier tag is found, the corresponding tagged word is added the post-modifier. The function terminates as soon the maximum post-modifier length of three is reached or if a tagged word does not qualify as post-modifier.
The resulting entities are then persisted along with their provenance information. All extracted pieces of information are depicted in table 4.4. With exception of the duration of the extraction as well as the on- and offsets of the pattern, each piece

| Name | Example |
|---|---|
| Pattern ID | p8a |
| Pay level domain (PLD) | example.org |
| List of instances | apple |
| List of instance tags | NN |
| List of classes | fruit |
| List of class tags | NN |
| Sentence | An apple is a fruit. |
| Onset of the pattern | 8 |
| Offset of the pattern | 14 |
| Duration of extraction process (ms) | 1000 |

Table 4.4: Extracted Data per Matched Pattern

of information is required in the following processing steps. The duration of the extraction is an artefact from the testing phase and was used to identify patterns with bad performance. It is measured per pattern and starts with the matching of the regular expression and ends as soon as the single entities of a tuple are extracted and stored. The on- and offsets are used to highlight the matched parts in a line. The information required for further processing steps starts with a unique identifier for the matched pattern, the so called Pattern ID. It is used to generate the attribute pattern spread, which describes how many different patterns found a certain tuple. Similarly the attribute PLD spread is generated from the PLD field. The core pieces of information are a list of classes and instances, which were extracted in front of and behind the pattern. From the two entity sets the cross product is built to generate pairs of classes and instances. For each word of the entities the corresponding POS tag is stored, as it is required for normalizing the single entities. Lastly, the entire sentence is stored as key component of the provenance information of the tuple.

One could assume, that building a tuple database with these results is straight forward. However, the aggregation requires large amounts of memory. In this experiment, with more than 2.1 billion extracted tuples, either an infrastructure with extensive memory is required or the aggregation is performed in multiple passes or the amount of data is reduced beforehand. The following section describes, how the amount of extracted data is transformed, without losing valuable information.

### 4.1.2 Removal of Duplicates

The removal of duplicates is the first step of the data transformation and is implemented to reduce the impact of spam on the aggregated attributes and to reduce the data volume for further processing steps. Spam in this context refers to frequently re-occurring phrases under the same pay-level-domain. Such re-occurring phrases are the result of standard templates, which are used to build up a web presence. These templates are used as component of each single web page of the web presence. In this research it is assumed, that enough valuable information is gathered, if the same sentence under the same pay-level-domain is analysed only once. The problem is that the single web pages of same PLD are spread over a number of archives. Thus spam cannot be detected completely on archive level, but has to be done after the extraction has finished. However, the with more than 1.3 billion extracted sentences with and average length of 179.6 characters it is not possible to load these into memory for a fast comparison. The solution is to perform the global duplicate removal using two passes over the extracted data. The first pass maps sentences into buckets, using a hash function, that makes sure that similar sentences, so called duplicate candidates, are grouped into the same bucket. The second pass then loads each single bucket into memory and removes duplicates. The decision for the proper hash function is not trivial. On the one hand it has to ensure, that buckets are not to big, otherwise the limits of the technical infrastructure would be exceeded. On the other hand the buckets should not be too small, otherwise file open and close operations of the second pass would increase the runtime. Due to the size and variety of input data, an exact determination of the best hash function is difficult. Therefore, multiple variations of the function are tested until a working solution is found. The resulting hash function takes the first three characters of a sentence as well as the length of the pay level domain. The concatenation of these two values is the result of the hash function. Although duplicates had already been removed on archive level, this processing step removed a large portion of the sentences and reduced the data volume significantly without losing valuable information.
Even with such a size reduction, it is not possible merge equal tuples in memory in one pass. A similar approach using buckets and a hash function is necessary. Therefore, tuples have to be normalized first, before they are sorted into buckets. Otherwise there would be no guarantee, that all equal tuples are in the same bucket. These normalization steps are described in the following section.

### 4.1.3 Tuple Normalization

At this stage only tuples will be transformed and therefore the sentences are split from the tuples, to reduce the duration of file reading and writing operations of each further processing step. The connection between a tuple and its sentence is kept using a unique ID. The unique ID is stored in both data sets, the sentences and the tuples. The tuple normalization is necessary, because the same entities are expressed in different ways in the corpus. The goal is to transform entities to a common string representation, so that equal entities have an equal string value. In this section the different causes for these *malformed* strings are presented along with a description, how these entities are transformed.

The reason for the first transformation step are the regex translations of patterns, as these allow optional symbols and commas and are not case sensitive. To highlight the need of the first step three examples are listed: *apple*, *'apple'* and *Apple*. All these extractions try to express the same term. However, they will not be merged, because they are not equal in terms of characters. So the first step is to transform all capital letters to lower case. Then all leading and trailing punctuations of each single word of an entity are removed. Second, all optional symbols such as apostrophes and quotation marks are removed. The apostrophes however, might also be part of a noun in the genitive form. A simple heuristic is used to identify genitive apostrophes, to prevent their removal. If a word contains an apostrophe followed the character *s* or if a word ends with *s* followed by an apostrophe. If one of these conditions is met, the apostrophe is not removed.

The next cleaning step is the removal of punctuations, which are *inside* a word of an entity. These are caused by the unclean nature of the corpus. Sentences on the web are not always written in proper grammar and the space behind a punctuation at the sentence end is simply left out. This has an adverse effect on the POS tagger and consequently the noun phrase extraction as demonstrated with the following example:

**Example 1** i like fruits such as apples.cause bob and alice told me.

After the *such as* pattern was found, the noun phrase detection is started. The POS tagger finds a punctuation inside the word *apples.cause* and therefore treats it as proper noun. To make it worse, the noun phrase detection adds bob, which is tagged as noun, to the existing noun phrase and then finds a co-ordination. This has the effect, that also *alice* is extracted as entity. This sentence would cause an extraction, in which *apples.cause bob* and *alice* are classified as *fruits*. Consequently the correction does not only involve the falsely punctuated entity, but also all entities that were detected afterwards. Therefore, if a punctuated entity is found, that

is not in the list of common abbreviations in Appendix F it is corrected. The correction treats entities detected *in front* of the pattern different to ones located *after* the pattern. For entities *after* the pattern all characters after the punctuation are removed. If the punctuated entity is *in front* of the pattern all characters before the punctuation are removed. If such an correction took place in a co-ordination, all entities that were detected afterwards are removed entirely, for both: Entities that are located in front or after the pattern. This might seem confusing for entities detected in front of the pattern, but one has to keep in mind, that these were acquired in reverse order. This punctuation removal might have the result that the new entity is a pronoun, which provides no useful information. Therefore it is once again necessary to remove tuples, in which one of the entities is a pronoun. The downfall of this approach is, that unknown abbreviations are split in half and terms, that naturally contain dots are also split. This mainly affects web pages. This step is acceptable, because it is not the goal of this experiment to extract websites, but rather domain independent noun phrases. And for most noun-phrases it can be assumed, that there is no punctuation inside one of its words. Furthermore, this procedure ensures, that POS tags for that entity are corrected as well, because entire words can be removed in this step.
After the removal of punctuated entities a length check is performed on the entire entity. 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 maximum length is the result of empirical evaluations. The initial allowed length in this step is set to 100 characters per entity. Entities longer than that are checked using the evaluation scheme of Ponzetto and Strube [33] and the result was 100 percent false extractions. The maximal length was then set to 50 characters. A check of 100 tuples with entities longer than that, shows that only 8 percent of these were correct. The correct ones are mostly technical or chemical terms, such as:
$\langle N-methyl4-(2-(2-(2-acetylaminopyridin-5-yl)-2-(R)-hydroxyethyl-N-tert-butyloxycarbonylamino)-ethoxy)-phenylacetamide, invention\rangle$ Eventually the maximum length was to 50 characters, as further tests with decreased length resulted in too much information loss.
The next data transformation step is concerned with the removal of inflected words. Such a transformation step is not uncommon before the matching or aggregation tuples or relations extracted from natural language texts [18] [33]. Thus lemmatization is applied on each word of a noun phrase. The result is the lemma of the word, which is defined in 5. The lemmatization is the main reason, why the POS tags of the entities have been stored. The used tool takes a word and its corresponding POS tag as input and returns the so called lemma of the word. The lemma is defined as:

**Definition 5** *"[...] the item which occurs at the beginning of a dictionary entry; more generally referred to as a headword. It is essentially an abstract representation, subsuming all the formal lexical variations which may apply: the verb walk, for example, subsumes walking, walks and walked." [10]*

With the lemmatized entities a further re-factoring 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. The issue regarding these entities is, that it is not easy to determine, which of these nouns is in a hyponymy relation with the entity behind the pattern. To demonstrate this problem three examples are provided below.

**Example 1** Works of artists such as Vivaldi.

**Example 2** Works of artists such as "The Four Seasons".

**Example 3** A variety of works such as "The Four Seasons".

In the first example 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. The second 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. The third example appears to be similar to the first one, because the last noun is in a hyponymy relation. However, the leading noun is a so called collective noun, as it refers to a group of entities [10]. The solution to resolve this ambiguity was determined empirically. Out of 100 tuples with multiple nouns, there are 21 starting with a collective noun. In all of these 21 instances the latter noun is the correct one. Hence, the decision is made to remove collective nouns and the subsequent preposition from the tuple. Although extensive lists of collective nouns are widely available [5], these are not used for performance reasons. In this research the list of collective nouns is determined empirically. A test set is used that consists of 1000 extractions, which contain two nouns separated by a preposition. Each occurring collective noun in that set is added to the list of excluded collective nouns, which is listed in Appendix G. With collective nouns removed, a further check of 100 entities is performed with a clear result. Either the first noun is correct or the

[5]Lists of collective nouns available at:
http://www.englishleap.com/grammar/collective-nouns
https://en.wiktionary.org/wiki/Appendix:Glossary_of_collective_nouns_by_collective_term
http://www.ojohaven.com/collectives/
http://speakspeak.com/resources/vocabulary-general-english/common-collective-nouns, accessed on 25.10.2015

tuple is wrong as a whole. Hence, if a noun phrase contains two nouns separated by a preposition the first noun can safely be set as head noun.
Using the normalization process, tuples that describe the same term are now mapped to the same string value. These string values are used in the following steps in order to aggregate tuples. This aggregation is performed in two steps. In the first step it is assured, that tuples have the same head noun combination. In the second aggregation step the modifications of the class and instance are checked and - if equal - aggregated. This *cascading* aggregation is described in detail in the following section, starting with the head noun extraction.

### 4.1.4 Tuple Aggregation

The tuple aggregation is the last processing step before tuples are stored. The aggregation not only reduces the data volume, but also generates three useful attributes. First, the occurrences of a tuple in the web corpus are measured, which is called the *frequency*. Second, the amount of distinct patterns, which identified the class instance pair are determined. This measurement is named the *pattern spread*. Eventually, the *PLD spread* is stored, which describes the amount of distinct PLDs that contained the hyponymy relation.
As stated before, the aggregation process takes the head noun as first aggregation criteria, which is at this stage determined using following heuristic. The head noun is the last noun in the first noun chain, or simply the first noun if no noun chains are present. One could argue, that nouns in the genitive form are an exception to this heuristic. However, these were already labelled as adjective in earlier stages. Thus nouns in genitive form do not interfere with the head noun detection.
To make the large amount of tuples unique, the same problems are faced as in the duplicate removal. Again two passes over the data are necessary to avoid long run times as well as too much memory consumption. The first pass uses a hash function to map potential merge candidates into buckets. In this case the hash function takes the length of the class and instance head nouns as input, to perform this mapping. In the second pass each bucket is read into memory and the tuples of each bucket are checked for parity. The parity check is performed in two steps. First, the head nouns of one tuple are compared to the head nouns of a second one. In case of identical head nouns, the tuples are merged regardless of their modifications in the first step. In case of a merge, the attributes for the *tuple head* are updated. Then the modifications are compared. If the modifications do not match, both modifications are stored. Both modifications are stored in a collection, which is associated with its tuple head. If these modifications do match, these are merged and during this process, the three attributes are updated. For this purpose the merged modification is then also stored in the collection, which is associated to the tuple head. This *cas-*

*cading* merge serves two purposes. On the one hand, the data volume is reduced, because the strings of the head nouns are only stored once for each unique class instance pair. On the other hand, this step not only generates attributes for modified entities, but also for the head nouns of a tuple.
During the aggregation a peculiar phenomenon becomes present. Although tuples, that originated from the same sentence under same pay-level-domain were removed during the duplicate removal, there were still occurrences in which a tuple occurred more than once under the same pay-level-domain and in the same sentence. This might seem impossible, however the explanation for this phenomenon is simple. This exception occurs, if a co-ordination contains the same entity more than once. As duplicate removal is performed on sentence level, these duplicates within the same sentences have so far gone unnoticed. This phenomenon is often observed for lists of ingredients, in which *sugar* or *salt* occurred more than once. If such a double entry in a coordination is detected, the tuple is ignored and the modification measurements are not updated.
After the aggregation of a bucket, the results are written onto disk and the next bucket is processed. After this step the tuples are in their final state and can now be stored in a database management system (DBMS). With more than $0.4$ billion unique tuples, a special storage concept is required to enable a reasonable query duration. This concept is described in the following section.

### 4.1.5 Data Storage

The main use case of the WTDB is to determine the most likely class and instance relations for a given entity. For this purpose it is necessary to acquire a large amount of related entities and evaluate, which of these relations are most likely to be correct. This use cases is enabled by a custom storage concept. The goal of this concept is to enable fast queries for a given class or instance term. Thus, the underlying data model has to be capable of returning large amounts of class or instance pairs for a given search term in a short amount of time. To achieve this an approach is taken, which groups and sorts pieces of data that are likely to be returned together in one single query.
The data model contains three types of tables. One for table for the provenance information, one for tuples sorted by classes and one for tuples sorted by instances. The sentence table contains sentences, the pay level domain of the sentence, as well as the ID that connects tuples with sentences.
The tuples are *sorted* into a number of tables using the head nouns of classes and instances using a hash function. This sorting increases query performance, because with a given head noun in the query, the table which contains all its related entities can be identified quickly. The same hash function that was used to sort and insert

the tuples is applied to the query term and returns the name of the table, which contains all the relations of an entity. The result is that the data is spread over a magnitude of comparatively small tables, instead of one very large table for all tuples. Due to the small size of the tables, the query can be executed faster. How this sorting is achieved is described in the following.
The entity table for a tuple is determined using the first two letters of a class head noun for class tables and the first two letters of the instance head noun for instances. If an entity contains less than two characters, the missing letters are replaced with zeros. As tables of classes and instances must not be mixed, the IDs of these tables start with the letter *i* for instances and the letter *c* for classes. The remainder of the ID are the two first letters of the inserted class or instance. To demonstrate the first insertion step the following example tuple is inserted: *(apple,fruit)*. As apple is the instance and starts with *ap* it is inserted in the table with the ID *iap*. The class fruit is inserted in the table *cfr*. With the corresponding tables identified the head nouns are stored as data field, along with its pattern spread, PLD-spread and frequency values. To enable a fast access to modifications of a *head noun tuple* without joins, all these are stored together together with their corresponding head tuple. Thus, these modifications are nested inside the data-object of the head tuple in form of a list.
As relationships are replaced with nested structures, it is not suitable to express the data model in the form of an entity relationship model. A custom model type is used to describe the nested structure, which is depicted in figure 4.2. The boxes in this figure are used to depict the hierarchical relations between the nested structures. If box *A* contains box *A*, then *B* is the super structure of *B*. The names of the boxes are depicted in **bold** letters. A super structure does not only contain sub structures, but also data fields, which are symbolized using a leading *dot*. Each data field consists of a unique name and a data type.

Besides the unique ID a head tuple contains a string value for class and instance heads, as well as aggregated attributes. As already stated the head tuple also contains a sub-structure in form of a list, which contains the modifications of its head nouns. A modification is stored with an unique ID and the string values of the instance pre-modifier *I-Pre-Modifier*, the instance post-modifier *I-Post-Mod*, the class pre-modifier *C-Post-Mod* and the class post-modifier *C-Post-Mod*. Besides the generated measurements for frequency, PLD-spread and pattern spread, there are three more data fields to enable more fine grained queries. All PLDs, all patterns and all provenance IDs *ProvIDs* are stored as string. These strings are the concatenations of the single PLDs, patterns and provenance IDs separated by semicolons.

With the storage of tuples and sentences in a DBMS the extraction work-flow is

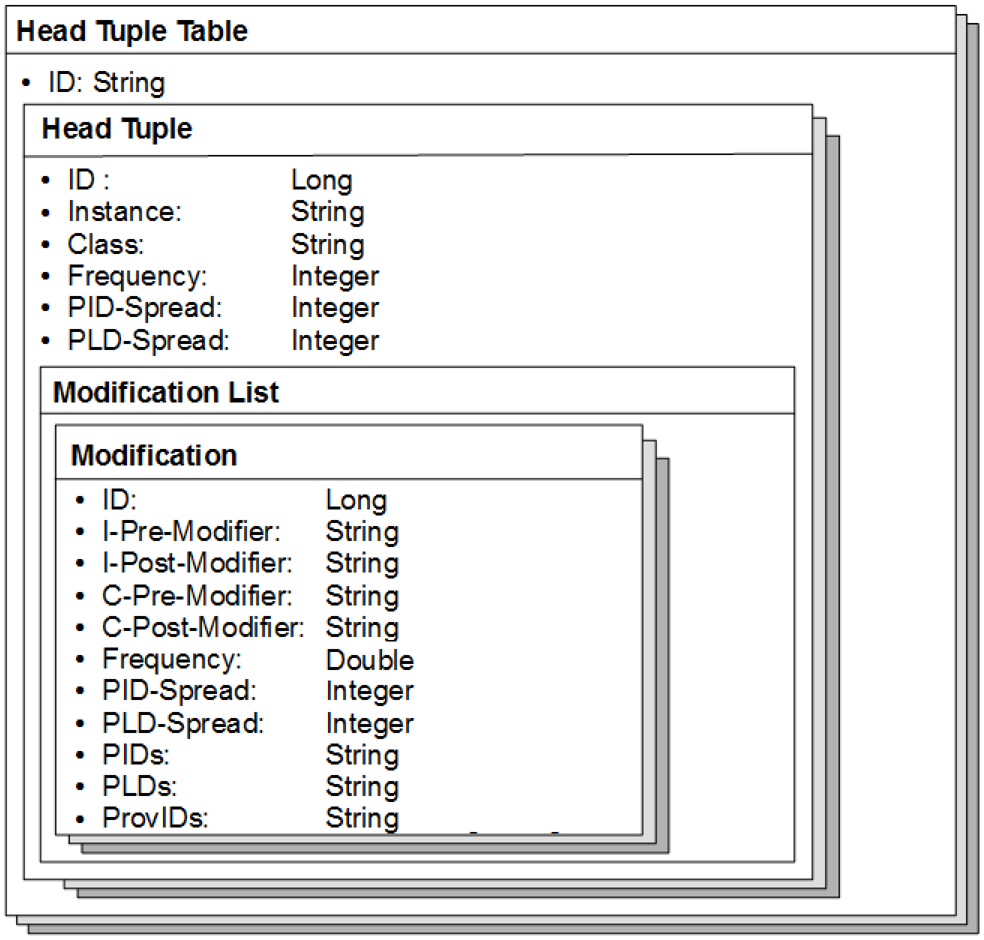

Figure 4.2: Data Model for Hyponymy Tuples

finished. The extraction started with a magnitude of archives that were constructed from a web crawl. The data in these archives is split and filtered before it is checked for possible hyponymy relations. If such a relation is detected, the single entities are extracted using natural language processing (NLP) techniques. Before these single entities are inserted into a database, they are normalized and aggregated. The single tuples along with the generated attributes are then stored in nested data structures. This processing pipeline requires different technical infrastructures and involves the selection and usage of a number of tools. These are described in the following section.

## 4.2 Technical Infrastructure

In this section the used infrastructure as well as the tools for the entire work-flow are described. The selection of the tools and technical equipment follows a basic principle. The larger the data volume, the more extensive is the technical equipment. In accordance to the work-flow the technical infrastructure is described starting with extraction of tuples from the Common Crawl.

### 4.2.1 Infrastructure for Data Extraction

It is not surprising, that the initial input data, the Common Crawl, is largest with over $168$ TB in size. This input data is distributed over a magnitude of archives. The processing steps take an average of $18$ minutes to extract tuples from one archive. With $38,609$ archives, the processing time alone with a single processor would take more than one year. Additionally, the mere download of the data is a problem as well, because the data volume exceeds the storage capacity of common hard drives. With these two problems in mind, an infrastructure is required, that on the one hand can store large amounts of data and also provides the processing power to perform the extraction. Such an infrastructure is provided by Amazon. Among others Amazon provides cloud computing services, which are called Amazon web Services (AWS) [6]. AWS can be classified as an infrastructure as a service (IaaS), which is defined below.

**Definition 6** *IaaS provides "processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications" [27].*

The selection of the Amazon infrastructure has a major advantage. The Common Crawl data is hosted by the AWS publicly. Although the data must still be down-

[6]An overview of AWS is available at https://aws.amazon.com/de/, accessed on 25.10.2015

| | |
|---|---|
| Instance ID | c1.xlarge |
| Operating System | Ubuntu 14.04 |
| Processor Architecture | 64 Bit |
| Virtual CPU Cores | 8 |
| Memory | 7 GB |
| Storage | 1.68 TB |

Table 4.5: System Specification for AWS EC2 Instances

loaded to the single machines, it is assumed that this download is faster, because the server hosting the data and the machines used to process it are provided by the same vendor. According to the IaaS model, AWS offers the rental of virtual machines, which can be used for the processing of data. Amazon offers a variety of different virtual machine configurations, so called *instance types*, with different usage fees and specifications. The machine selection has to be economic. Hence, it is paramount to select a machine that is able to accomplish the task at the lowest cost level. The bottleneck of the tuple extraction are the processor cores and not memory capacity. Thus an instance type is selected, that offers as much processing power as possible. With this decision criteria in mind, the instance type *c1.xlarge* [7] is chosen, because it has eight processing cores and a relatively low memory capacity. An overview of the system specification of one instance is provided in table 4.5. With 100 of these machines the processing time is reduced to 22 hours.

The archives of the Common Crawl as well as the extraction program, now have to be distributed to these 100 instances in order to extract tuples. The extracted data along with meta data about the extraction process has to be stored at a central location. This complex routing process regarding the input and output data of the tuple extraction is completely handled by a framework developed by Meusel et al.[8]. The so called *Web Data Commons Extraction Framework* is specifically designed to operate on the AWS infrastructure. Its functionality is depicted in figure 4.3 and described in the following paragraph.

The figure contains rectangles, which symbolise a machine in the Amazon infrastructure. These machines are connected with arrows, which depict the flow of data or messages. One of the machines is the master node. The user can access the

[7] Overview of instance types available at https://aws.amazon.com/ec2/instance-types/, accessed on 25.10.2015

[8] Download and implementation instructions available at http://webdatacommons.org/framework/, accessed on 25.10.2015

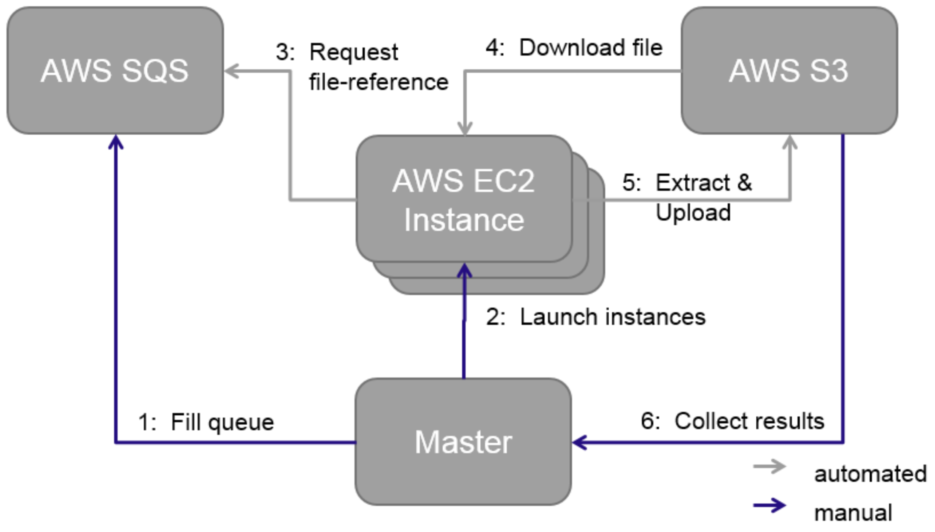

Figure 4.3: Web Data Commons Extraction Framework. Adapted from [29]

master node using a command line interface to steer and configure the extraction process. Using its command line interface, a job queue is filled with archives of the Common Crawl. This job queue managed using the Amazon Simple Queue Service (SQS), which is a message passing mechanism that distributes the workload to the single instances [41]. After the instances are started using the command line interface of the master node, each instance contacts the SQS to request a file-reference, which points to one of the archives of the Common Crawl. After such an instance successfully requested a file-reference, it downloads the corresponding Common Crawl archive from the Amazon Simple Storage Service (S3) and starts to extract hyponymy relations. After an archive has been processed, the extracted data as well as meta data about the extraction is stored using the S3 service. Among other the meta data provides information about errors and exceptions during the extraction as well as statistics of the extraction of each single archive. The meta data was used during the testing phase to measure the effect of the algorithms on the amount of errors and the extraction duration. After each queued archive is processed, the extracted data and meta data can be manually downloaded from the S3 using the command line interface.
Overall, this framework takes care of the acquisition and distribution of input data, as well as the storage of the extracted output. The framework also deploys the extraction algorithm on each of the instances. The advantage of this framework is,

that only the extraction algorithm has to be implemented as one Java class. The disadvantage is, that this Java class - including all its imported third party tools - has to compatible with the framework, which is written in Java version 7. Thus not only the code of the extraction, but also integrated third party tools for the extraction algorithm have to be compatible with the framework.
The first third party tool is named *Stanford CoreNLP* in version 3.4.1 [9], which contains an entire set of natural language analysis tools and is the latest version compatible with Java version of the framework. These tools are used for POS tagging and later on for tuple normalization. The first reason to select this tool is its compliance with the programming language of the framework. Second, the tool is available under general public license, which allows its use for this research. Lastly, the POS tagging functionality of this external tool is state-of-the-art in terms of accuracy [10]. The POS tagging uses an external model to determine the part-of-speech tags. Each language requires its own model. For the English language, there are two different models available, which differ in terms of accuracy and runtime. The model with lower runtime and accuracy - named *english-left3words-distsim.tagger* - is chosen for this experiment. Although the second model has a slightly higher accuracy, the runtime increase can not be justified. Even if a word in a sentence is falsely tagged, this should not effect the aggregated results dramatically. With the high recall one can expect, that in most of the sentences the entity is been tagged correctly.
The second external tool used for tuple extraction is required to identify the PLD of a web page. Such a functionality is provided by *Guava* in version 13.0.1 [11], which is maintained by Google. This library contains a list of top-level-domains, which can be used to detect the pay-level-domains of an URL. The described infrastructure extracts tuples from $38,609$ archives in about 22 hours. The extracted data has a compressed size of $115$ GB and is only a fraction of the initial input data. This size reduction enables the usage of a less extensive infrastructure for further processing steps.

### 4.2.2 Infrastructure for Data Transformation

Beginning with the removal of duplicates, a new and less extensive infrastructure is used. The removal of duplicates requires the temporary storage of sentences and

[9]Available at http://nlp.stanford.edu/software/stanford-corenlp-full-2014-08-27.zip, accessed on 25.10.2015

[10]Evaluation of POS tagging tools available at http://aclweb.org/aclwiki/index.php?title=POS_Tagging_%28State_of_the_art%29, accessed on 25.10.2015

[11]Available at http://search.maven.org/remotecontent?filepath=com/google/guava/guava/13.0.1/guava-13.0.1.jar, accessed on 25.10.2015

| Property | Value |
|---|---|
| Operating System | Ubuntu 14.04 |
| Processor Architecture | 64 Bit |
| Virtual CPU Cores | 1 |
| Memory | 50 GB |
| Storage | 500 GB |

Table 4.6: System Specification for Data Transformation

PLDs. Even with the approach described in section 4.1.2, a considerable amount of data has to be stored in memory. Such an increased memory capacity is also required during the tuple aggregation process. Consequently, the machine configuration with the highest possible memory capacity is chosen, which is 50 GB. The output of a data transformation task serves as input for the next task. Although input data can be deleted after it has been processed, the decision is made to keep the data of each step. On the one hand, it is useful to have to ability to look into the results of each processing step. On the other hand, it is not necessary to restart the entire processing pipeline again, if one of the later processing steps fails. With the data of each step stored, the transformation process can be resumed at any point. Thus the required storage capacity has to be estimated. The input data from the tuple extraction has a compressed size of 115 GB. In the worst case the extraction contains zero duplicates, which would result in another 115 GB of data. Since the data is re-arranged into buckets during this step, these buckets store a copy of the input data. The removal of duplicates alone, requires a capacity of 345 GB in the worst case. In the next steps only tuples are stored and transformed, therefore only a fraction of the size is expected as output of the following tasks. In this experiment it is estimated that the sentence of a tuple takes four times more storage capacity, than only the tuple. As a result, the output data of the following task - the tuple normalization - is estimated to be around 30 GB. Since the tuple aggregation requires a pre-selection of aggregation candidates, the 30 GB input data has to be copied and sorted into buckets, before tuple aggregation. In the worst case there are no tuples to be aggregated and the input is equal to the output, which in turn results in another 30 GB output data. Thus the tuple data is replicated three times in the worst case, which sums up to 90 GB of required storage capacity. Overall, it is estimated that in the worst case 435 GB storage capacity are necessary for the data transformation. With these estimations the decision is made to select a 500 GB hard drive as storage medium for these tasks. These hardware specifications are sufficient for the data transformation tasks and are summarized in table 4.6.

In terms of software, the only requirements are an operating system with Java Version 7 installed. The code which executes the data transformation uses one already known external library: The *Stanford CoreNLP* in version 3.4.1, which is used to lemmatize words during the normalization of tuples. The reason to use the same NLP library in the data normalization step is simple. The set of POS tags used during the extraction must be equal to the one used for lemmatization. If another tool would be used for lemmatization, all sentences would have to be POS tagged again. With this infrastructure it is possible to transform the data within two days. After this processing the data is inserted into a DBMS. The infrastructure for inserting and hosting the data in a DBMS is described in the next section.

### 4.2.3 A DBMS for Tuple Storage

In this section the infrastructure for the tuple storage is described. The goal of this infrastructure is to enable fast queries for classes and/or instances. The extracted tuples have a compressed size of about 15 GB and the provenance information requires 46 GB. The mere size of the extracted data does not allow an in-memory storage. Even if only the tuples are loaded into memory, they would at least occupy 15 GB permanently. Using approaches like dictionary encoding to reduce redundancy of string values are expected to be of little help in this case. As there are more than 200 million unique entities, each of them would require an entry in the dictionary. A dictionary of such a size will also require large amounts of memory. Furthermore, besides the actual tuple data, there are two dictionaries necessary. One to translate the query terms into numbers and one to translate query results back into words. For these reasons, an in-memory approach was not tested in this work.
The aspects stated before entail the usage of a DBMS, that stores data on disk. To select the proper system, a further classification of DBMS is used. A DBMS can classified into relational and non-relational DBMS. In this work one has to model N:M relations, which is best modelled using a relational approach [37]. This is due to that fact that an instance can be related to multiple classes. For example an *apple* can be the instance of *fruit*, as well as *company*. On the other hand a class, such as *company*, contains multiple instances, for example *Apple* and *Google*. In the relational approach a N:M relation is usually modelled using three tables. One table for each entity type, which store attributes of the entities. The third table is used to store the unique IDs of the entity tables to model the relation. If two entities are in a relation, their corresponding unique IDs are stored in one row of the relation table. If a query involves attributes from both of the entity tables, the relation table is used to join these two data sets. However, such a join is rather costly with more than 400 million unique class-instance combinations. The goal of the database is

to enable the fast search for all classes given an instance name, and vice versa. Consequently each query requires the join of class and instance tables, which in turn requires the processing of a table containing more than 400 million relations. Since this would lead to highly increased durations, a non relational database is chosen.
Non-relational DBMS, also called NoSQL DBMS, have two important advantages compared to relational ones. First they are able to read and write data quickly [23]. As this experiment handles large amounts of data, this is a significant advantage during the insertion of data. Also the fast reading is important for the fast query execution, if one considers the result of queries for general concepts like *thing* will require reading operations for a large number for instances. The second advantage is, that this type of DBMS is specifically designed for mass storage [23]. With these advantages in mind, the decision is made for a non relational DBMS. For the proper selection of the non-relational DBMS a closer look at the data is necessary. As depicted in 4.2, a single tuple contains more than a class and instance value, with its modifications. There are also generated measures, such as frequency, as well as lists with information about the tuple provenance. One type of non-relational DBMS are document databases. The focus of these is on fast query performance and big data storage [23]. In this context an entry in such a database is called document. These documents are persisted using the Binary JavaScript Object Notation (BSON) format and are capable of holding complex data structures such as lists [47]. Since these document databases allow the storage of big objects and focus on fast query performance, they fulfil all requirements for this experiment. There is a magnitude of document database implementations available [6]. For this experiment MongoDB in version 3.0 [12] is chosen, since its license is free of charge and has a Java application programming interface (API). Although other document databases like OrientDB or OpenLink Virtuoso have similar characteristics, the decision is made for MongoDB due to its popularity. With its widespread usage there is a magnitude of support for developers available online, which was the decisive criteria for the selection. The support facilitated the development of both the insert and the query operations. The query operations can executed using an API, which is described in the following section.

## 4.3 API for the WTDB

So far, the extracted data is stored in a document database and single entries can be retrieved using queries in the MongoDB specific query syntax. However, it is a time

[12]Available at https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-amazon-3.0.7.tgz, accessed on 25.10.2015

| Name | Type | Example | Description |
|---|---|---|---|
| instanceHead | String | "apple" | The head noun of the instance |
| clazzHead | String | "fruit" | The head noun of the class |
| iPreModifier | String | "red" | The pre-modifier of the instance |
| iPostModifier | String | "in Spain" | The post-modifier of the instance |
| cPreModifier | String | "healthy" | The pre-modifier of the class |
| cPostModifier | String | "for people" | The post-modifier of the class |

Table 4.7: API Parameters to Specify Classes and Instances

consuming task to learn a new query language. Additionally, if any possible query is allowed, there can be no guarantee, that the infrastructure is able to process the query. For these reasons, an application programming interface (API) is provided. The API is developed in Java, because this programming language is platform independent. Another reason to use Java is its wide-spread usage. This decision ensures, that the API can easily be used by many people on all platforms, that are able to run a MongoDB server.
The API provides the users with methods to query the provenance information and hyponymy tuples. If one of these methods is called, the API establishes a connection to the database, translates the user specified parameters into a query and executes it. The API is implemented in a way that ensures, that it does not consume a lot of memory. This is achieved by returning the single results of a query directly after their detection. With a maximum size of 16 MB for a stored tuple [22], the memory consumption of the API is diminutive.
The main use case of the API is to detect all class or instance relations of an entity. The words of an entity can be specified by head noun or by its modifiers. Since the tuples in the database are grouped by their head nouns, the specification of the head noun parameter increases query performance drastically. Since all tuples with the same head noun for that entity are grouped in the same table, only one table has to be queried for results. If the user does not want to specify a head noun, a wildcard can be used as input parameter. For the head noun and all other parameters of the type string, the common asterisk symbol can be used as wildcard. The API parameters to specify the words of an entity are summarized in table 4.7.

For a more fine grained analysis, the API allows the user to restrict the results depending on their provenance information. This includes the specification of patterns and PLDs, as depicted in table 4.8. The patterns differ in precision and recall. Thus the API enables the user to select the patterns according to his pref-

| Name | Type | Example | Description |
|---|---|---|---|
| pids | String[] | ["p1", "p2"] | The pattern IDs of a tuple. |
| plds | String[] | ["google.com"] | The pay-level-domains of a tuple |
| strict | Boolean | true | Specifies, how the lists of $plds$ or $pids$ are handled [13] |

Table 4.8: API Parameters to Specify Domains and Patterns

erence. Similarly the API has to allow a filtering of PLDs. The different PLDs cover different domains and contain unstructured data of different quality. These API parameters can be specified using arrays of string values. The user is given the additional option to specify, if a tuple has to cover each single specified pattern and PLD or just one of them. For this purpose the boolean parameter $strict$ is provided. If this parameter is set to $true$, then each single pattern or PLD in the list has to be present in the provenance information of a tuple. On the other hand, if set to false, only one single pattern or PLD of the lists has to be present for a tuple. If a user does not want to specify any PLDs or patterns for the query, the parameters can be set as an empty array.

Lastly, a user is able to set value ranges for each of the aggregated attributes. The API allows the specification of minimum and maximum values for the frequency, the pattern spread and the PLD spread of a tuple. Similar to all the other query parameters, the user is able to use wildcards for the aggregated attributes. As per definition the values of the attributes can not be smaller than $1$, the wildcard for these attributes is the value $0$. It is important to note, that the API does not perform a validity check of these attributes. For example, the user is able to specify a minimum and maximum frequency, where the minimum value is greater than the maximum. The API will simply execute the query and return zero results. An overview of the API parameters, to specify the aggregated attributes is provided in table 4.9.

While the API executes the query, the single results are returned one after another. If each single parameter is initialized as wildcard, every single tuple stored in the database will be returned. The result contains all data fields of a tuple, as described in figure 4.2. Often times the user wants to find out, where a certain result was extracted. For this purpose the API allows the user to query the provenance information of an extracted tuple. For this purpose the API provides a method, in which the user can specify a provenance ID to acquire the sentence and the PLD of an extracted tuple. If the method is called without any provenance ID specified,

| Name | Type | Example | Description |
|---|---|---|---|
| minFrequency | Double | 1 | Specifies the minimum frequency of a tuple |
| maxFrequency | Double | 205556 | Specifies the maximum frequency of a tuple |
| minPidSpread | Integer | 1 | Specifies the minimum pattern spread of a tuple |
| maxPidspread | Integer | 47 | Specifies the maximum pattern spread of a tuple |
| minPldSpread | Integer | 1 | Specifies the minimum pay-level-domain spread of a tuple |
| maxPldSpread | Integer | 15658 | Specifies the maximum pay-level-domain spread of a tuple |

Table 4.9: API Parameters to Specify Aggregated Attributes

each single piece of provenance information is returned.
Overall, this API serves two use cases. First the export of data. If a user executes both methods with only wildcards as parameters, the entire content of the database can be returned. Second, the API enables the user to execute parametrized queries to find information about the class or instance relationships of an entity. As this tuple information was acquired in an unsupervised fashion, it is now interesting to see how much information is gathered and to which degree the extracted information is correct. These quality criteria are discussed in following chapter.

# Chapter 5

# Experimental Results

For each step of the presented work, this chapter provides an analysis of the results. The following sections describe the statistics about the tuples, about the created database and finally a critic analysis about the tuple attribute values. The former also includes the error analysis and provides solutions for all these errors.

## 5.1 Statistics of the Extraction and Data Tranformation

The initial input of the extraction process is the Common Crawl corpus. This input data has a size of 168 TB, which are distributed over $38,609$ archives. As imposed by the Amazon infrastructure and required by the framework developed by Meusel et al. [29] , each archive has to be processed within a user defined time frame. If this time frame is exceeded, the process which crawls the single archive is terminated. This mechanism ensures that instances do not spend too much time on a single archive with ill-formed data, which in turn would increase the costs and the duration of the extraction. This time limit was exceeded for $626$ archives, which is approximately $1.62$ percent of the input data. This failure rate highlights the issues of such a large and heterogeneous input corpus. Even with a number of successful pre-tests without a single error one cannot be sure, that the the algorithms can handle the entire data without errors. However, a closer look at the meta data of of the extraction reveals, that the errors occurred within certain time frames. In one time frame of less then $90$ minutes $245$ out of the $626$ errors occurred. This heterogeneous distribution of errors over the time frame of $22$ hours might indicate, that these errors are not caused by the code, but rather by the used services of Amazon. However, since the number of failed archives is in a acceptable frame, the decision is made to not further research the single error causes. The entire extraction was performed in $22$ hours and the compressed

| | |
|---|---:|
| Input Size | 168 TB |
| Input Archives | 38609 |
| Processed Archives | 37.983 |
| Failed Archives | 626 |
| Size of Extraction | 115 GB |
| Duration of Extraction | 22h |
| Extracted Tuples | 2,121,495,294 |
| Extracted Sentences | 1,336,115,725 |
| Used Patterns | 58 |

Table 5.1: Quantitative Results of the Tuple Extraction

output has a size of 115 GB. The results of the extraction, which applied a total of 58 patters on the corpus, are summarized in table 5.1. The extraction resulted in over 2.1 billion extracted hyponymy tuples. These tuples originate from *only* 1.3 billion sentences. These two values show, that there is a significant amount of sentences in the corpus, which contain more than one tuple. Multiple extractions per sentence are either caused by co-ordinations or multiple pattern matches per sentence.

The large amounts of extracted sentences and the contained tuples are reduced during the duplicate removal. As described in section 4.1.2, a sentence must only be extracted once per PLD. The quantitative results of the duplicate removal are depicted in table 5.2. From the extracted 1.3 billion sentences almost 0.9 billion are duplicates. The results show, that the textual information on single PLDs is often duplicated. A natural cause of such a duplication is the usage of templates for website design. If such a template is used on every web page under the the same PLD and contains a pattern, it is extracted for each single website. The results highlight the need of a better duplicate removal during the extraction phase, because almost 70 percent of the extracted data contains duplicate information. This data overhead increases costs and duration of the extraction because the matching of patterns and the extraction of single noun phrases is performed multiple times for the same sentence. Thus removing all duplicates *globally* for all archives, decreases the runtime and the costs of the extraction. Additionally, the duration for downloading the data is reduced, which should not be underestimated for such a large scale extraction.
After the duplicates are removed, the tuple normalization is executed. During this process tuples are transformed and become invalid in certain cases, as described in section 4.1.3. There are more than 55 million invalid extractions. There are two causes for the removal of tuples. On the one hand, extracted entities longer

| | |
|---|---:|
| Duplicate Sentences | 879,418,399 |
| Duplicate Tuples | 1,414,136,985 |
| Invalid Tuples | 55,547,285 |
| Remaining Tuples | 651,811,024 |

Table 5.2: Quantitative Results of the Duplicate Removal and Tuple Normalization

than 50 characters are removed. On the other hand, if an entity was punctuated and not identified as abbreviation, further removals are performed. The remaining 0.6 billion tuples are now used in the aggregation process to generate attributes for the tuples using the frequency and diversity of their provenance information. The results of the aggregation process are unique tuples, which are stored in a DBMS. The statistics describing the WTDB are described in the following section.

## 5.2 Statistics of the WTDB

The final resource is a MongoDB database and has a physical size of 645 GB. Its physical size is larger than the extracted data, not only because it is uncompressed. Also the highly redundant data model attributes to such an increase in size. Since the tuple objects have sizes of up to 16 MB it is no surprise that 0.4 billion tuple objects require such an amount of physical storage. There are more sentences stored in the WTDB than unique tuples, because the sentences are neither normalized or aggregated. The sentences are stored as they are directly after the removal of duplicate sentences per PLD. The decision was made to keep these results for further analysis. This magnitude of tuples consists of almost 107 million unique instances and about 121 million unique classes. Overall, the WTDB contains more than 200 million unique entities. Compared to other large scale web extractions, the results are far more comprehensive. As stated by Wu et al. [45] Probase, was in 2012 the most comprehensive taxonomy. To evaluate the success of the high recall guideline, it is suitable to compare the extraction of Wu et al. [45] as benchmark. Probase contains $2,653,872$ distinct entities and $20,757,545$ unique tuples in total. Compared to this work, the results of Probase appear to be diminutive. The amount of unique tuples stored in the WTDB is approximately 10 times larger and the amount of unique tuples of the WTDB beats Probase by a factor of approximately 20. However, the quality of the extracted tuples in the WTDB has not been evaluated thoroughly. To create such a taxonomy as Probase, further cleaning steps are likely, which in turn will decrease the amount of entities and tu-

| | |
|---|---:|
| Physical Size of Database | 645 GB |
| Amount of Sentences | 456,697,326 |
| Amount of Unique Tuples | 401,150,041 |
| Amount of Unique Entities | 212,155,722 |
| Unique Instances | 107,691,822 |
| Unique Classes | 120,992,248 |
| Intersection of Classes and Instances | 16,499,109 |

Table 5.3: Quantitative Results of the WTDB

ples. Nevertheless these statistics indicate, that the WTDB is a decent foundation to create a more comprehensive taxonomy as Probase. To create such a taxonomy, the intersection of classes and instances must not be an empty set. Otherwise, the hierarchy of the created taxonomy has a depth of $1$. The intersection of classes and instances has a size of $16.5$ million. For this measure tuples were excluded, that have an equal class and instance, because these would result in an intersection per definition. Such a result indicates, that the extracted data can be used to create a taxonomy. The statistics of the WTDB are summarized in table 5.3.

As second contribution of this thesis, the aggregation of tuples provided three attributes: Frequency, pattern spread and PLD spread. These attributes are gathered after extractions from the same sentence under the same PLD were removed as duplicates. The minimum observed frequency is $1$. This frequency value entails, that the minimum values for pattern spread and PLD spread must also be $1$. For this reason the minimum values are not included in table 5.4, which summarizes statistics about the generated attribute values. The highest observed frequency is about $205,556$, which means that this unique class instance combination was found in more than $200,000$ different sentences. On average a tuple has a frequency of $1.5142$. This average value shows, that there must be a very large amounts of tuples with a frequency of $1$. To be exact, the WTDB contains $341,306,245$ tuples with lowest possible frequency.

The patterns themselves are not evaluated in terms of precision and recall for two reasons. On the one hand, such an evaluation requires a sample of tuples for each pattern. As already mentioned it is difficult to select a subset for each pattern that represents the different attribute values. On the other hand, a similar evaluation was already conducted before the extraction was started to select the patterns.

For this magnitude of tuples a custom storage concept was developed and implemented in a so called document database. An API tailored to the storage concept

| | |
|---|---:|
| Average Pattern Spread | 1.0576 |
| Average Pay-Level-Domain spread | 1.3291 |
| Average Frequency | 1.5142 |
| Maximum Pattern Spread | 52 |
| Maximum Pay-Level-Domain Spread | 10,096 |
| Maximum Frequency | 205,556 |

Table 5.4: Statistics of Generated Attributes

is implemented to provide fast queries for classes and instances. The access speed is now estimated by gathering statistics about the query execution. The decision is made not to execute random queries, not only because of the diversity in tuple size. The intended purpose of the WTDB is to enable users to search for all related entities of an instance or a class. Hence, the duration of the query execution is often influenced be the amount of related entities. This leads once again to the question, how to choose queries for an representative test of the database. As there is an infinite amount of possible queries, a strategy is required to select suited ones. For this purpose queries are created, which not only return large objects, but also a large amount of query results. This can be seen as a stress-test, which tries to provoke the largest possible query durations. The queries are generated from the top-1000 tuples according to PLD spread. These are per nature rather large, because each single PLD is stored within this database object. As the narrative evaluation of this attribute in section 5.3.3 shows, the most spread tuples are expected to have large amounts of related entities. The reason for the large amount of relations is the popularity of the entities. Based on this information, the assumption is made that this list of tuples, will translate into queries that return large amounts of results. With classes such as *thing* or *people* this assumption this appears to be a reasonable. The list of 1000 tuples was translated into three sets of queries, each containing 100 queries. The first set of queries has class and instance terms specified, including their modifications. The goal is to test the performance of queries which return one single and comparatively large object. In the next query sets either only the class terms are specified and the instance terms *wildcarded* or vice versa. During the query generation it is assured that queries are unique, because some entities occur more than once in that list. The results are summarized in table 5.5.

In this test the duration is measured per query set. The duration measurement starts with the first entry in the set and stops, when the last entry has been returned. Then the total duration per set and the amount of returned tuples are measured. The results show that queries, in which classes and instances are both specified are

| Specified Attributes | Duration (ms) | Returned Results |
|---|---:|---:|
| Class and Instance | 1,072,281 | 100 |
| Class | 1,935,572 | 8,462,925 |
| Instance | 1,405,475 | 8,147,608 |

Table 5.5: Statistics about the Query Execution

faster than the ones in which only one of the entities is specified. The query duration is lower, because the DBMS has to read only one single entry. The reason, why the remaining two query sets are only slightly slower, is no surprise. The data model was specifically created to return large amounts of tuples the share either the same class or the same instance.
The average duration of $10.7$ seconds for queries, which return only one single tuple is by no means a good result. Such a query duration harms the usability of the WTDB. However, this query type is not the intended use case of the WTDB.
The query performance of the *wildcarded* queries appears to be unacceptable at first glance. An average of about $19.36$ seconds per query with only class terms specified harms the usability of the WTDB. However, it is important to note that each single query returns an average of $84.629, 25$ related entities. Due to the API implementation the results are returned one after another. Thus each of these queries returns on average $4.371$ objects per second to the output of choice. For the purpose of extracting a large amount of related entities in a short time, in order to evaluate the most *correct* relations, such an output is absolutely sufficient. The queries, in which only the instance terms are specified are even faster. The average execution time is $14.05$ seconds and has an output of approximately $81, 476.08$ results. Thus, these queries return $5.799$ tuples per second. There are two explanations why the queries using a specified class are slower. On the one hand, there are more classes than instances stored in the WTDB. Thus it can be assumed, that the DBMS requires more time to locate the entry in the slightly larger class tables. On the other hand, the query duration is dependent on the position of the entity in the modification list. This list is iterated until the exact combination of modifications is found, which is specified in the query. Thus queries are faster for modified entities, if these are located at the beginning of this list. The used queries are depicted in Appendix J. As the query duration is dependent on the used system, which hosts the MongoDB server, the system specification used for the query execution is provided below.

| | |
|---|---|
| Operating System | Windows 7 |
| CPU Architecture | 64 Bit |
| CPU cores | 4 |
| Memory | 16 GB |
| Physical Storage | 3 TB |

Table 5.6: System Specification used for Query Execution

## 5.3 Narrative Evaluation

The mere size and the content of the WTDB makes a evaluation difficult. Due to the variety of concepts present in the web corpus, a researcher can not tag the relations of such a concept as true and false in a matter of seconds. Often times the researcher himself, has to look for definitions, classifications or sub-concepts to make a justified true or false decision. With many statements on the web being highly opinionated and colloquial, the evaluation becomes even more time consuming. An example is the tuple $\langle Obama, commie\,terrorist\rangle$. First, the researcher has to find out the meaning of colloquial terms. On the other hand the researcher might not share the opinion of the creator(s) of the hyponymy relation and therefore requires more time to find a sense, in which the tuple is correct. Thus an evaluation of large amounts of samples is not feasible. A strategy has to be found to acquire a small sample of the data, that still represents the content of the database. A completely random sample will not represent the WTDB, because it will be biased. $85$ percent of tuples have a frequency of $1$ and therefore it is likely that most of the entries in a random subset, have the same attribute values. On the other hand it is not to select a subset that represents any attribute value. As intended from the beginning the value ranges are large. The attributes have value range between $52$ for pattern spread and $205556$ for frequency. Consequently such a sample would have a size, that is too large to evaluate in a reasonable time frame. Another way to extract a sample is to specify a certain domain, e.g. religion, and collect a list concepts used in this domain from literature. Although this approach sounds promising it has a number of downfalls. Due to the nature of the corpus, the selection of a single domain is biased. Some domains on the web are broadly discussed, such as pop-culture, whereas some others, for example philately, are merely mentioned. Additionally, with the abundance of different domains in the corpus it is difficult to make a justified decision.

The conclusion is, that there is no suitable strategy to select a meaningful subset from the corpus, that represents the data stored in the database and at the same time can be evaluated within a acceptable duration. For this reason, an analysis is conducted, which aims to find errors of and improvements for the data extraction and transformation process. For this purpose the characteristics of tuples are analysed, which have the largest values for each generated attribute. These attributes are generated with the assumption, that large attribute values indicate tuple correctness. The goal is to find common characteristics of wrong extractions in these top-k lists, to infer improvements for the extraction and transformation process. It is important to note, that the extracted top-k lists do not include tuples, which have an equal value for class and instance.

### 5.3.1 Evaluation of Top25 Tuples by Frequency

A high frequency of a tuple is caused, if the same class instance pair occurs in a lot of different sentences under different PLDs. It is peculiar, that all tuples in the top-frequency tuples are spread only over a handful of PLDs. This means, that these tuples are used by only a few websites, in a vast amount of different sentences. It is hard to believe, that one website has over $200,000$ different hand written sentences, which contain the same tuple. A more plausible explanation for these attribute combinations is that these sentences have been automatically generated or belong to a standard template used for each web page of a PLD. An evidence for the presence of generated content or templates is, that all these tuples were found with one single pattern. Although not all of these extractions are wrong, they should not be extracted that frequently. The goal of this research is to extract hyponymy relations from natural language texts and not from automatically generated content. The heavy increase of extracted data cannot be justified, by such a low information gain from automatically generated content. Thus, a solution has to be found to filter these tuples before they are aggregated. Since these tuples have not been removed during the duplicate removal, there must be parts included in the sentences, which make it different from others. This part is a unique identifier, which is included in these sentences. The websites hosting these tuples are mostly large resources for content, in which each single item is displayed on a single web page. The problem is, that each single item has a unique ID followed by an automatically generated sentence. A possible solution to reduce the impact of automatically generated content can be implemented in the duplicate removal. Instead of checking for exactly the same sentences, the method should rather use similarity measures to compare sentences. If this measure is above a certain threshold, the sentence has to be tagged as generated content and should be removed.
Furthermore, this top-k list reveals downfalls of the Common Crawl archive selection. The archives of type WET were selected, because these contained only the

textual representation of web pages with html tags removed. However, some html tags are often used for text formatting, such as $< \&nbsp; >$. If a creator of a website uses these tags, instead of white- spaces, the structure of the extracted sentence is not represented correctly in the WET archives. If these tags are removed, the result is, that previously separated words are concatenated. Thus the POS tagging is not able to extract the original noun phrases. This phenomenon is visible in the in instances $conditionscontact\ usbuilt\ by\ orange$ or $rights\ reservedabout$. Since these errors are present in the WET archives, the solution for this problem would be to choose the WAT archive type. This archive type contains the html tags and the textual data of a web page. Instead of removing all tags from the website to acquire its textual representation, certain html tags used for text formatting can be replaced by their corresponding characters. If such a solution is economically feasible is another question. The size of the input data and the complexity of algorithms will both increase with the presented solution.
Moreover, some of the most frequent tuples contain invalid noun phrases. For example the tuples $\langle below, map\rangle$ and $\langle send\ there, error\rangle$ do not contain a noun in the instance. This is caused by the rather heterogeneous corpus in combination with the POS tagger. Many words are tagged wrong, because the web data is different from the corpus, which was used to train the POS-tagger. Furthermore, the POS-Tagging tool was configured for a fast executions and not for high precision. To reduce the amount these errors, the POS tagging has to be more accurate, which either requires more sophisticated tools or a configuration that focusses on precision. However, the performance of the extraction will suffer, because the POS tagging will require more time to finish.
A further error of the extracted data becomes visible when looking at the tuples $\langle worldcat, linked\ data\ resources\rangle$ and $\langle worldcat, linked\ data\ resource\rangle$.
Although lemmatization was performed during the tuple normalization, the noun $resources$ is still in plural form. The only explanation of this error is, that the word resource is POS tagged differently in both cases. The word $resource$ can either be used as a verb or as a noun. Thus the lemmatization will result in two different outputs. The solution for this problem the same as above. A more sophisticated POS tagger or a POS tagger configuration that focusses on precision.
Lastly, an expected error is present in the most frequent tuples. During the normalization of tuples, entities are corrected, that contain a punctuation *inside* one of the words. As every website contains a full-stop surrounded by characters, every single website is affected by this cleaning step. In the case of the tuple $\langle year\ of\ archive\ content, com\rangle$ all the characters in front of the full-stop have been removed. To solve this problem, websites and other entities, which naturally contain a punctuation, require special treatment during the cleaning phase.
Overall, only 6 of the 25 tuples can be classified as correct. This is an indication,

that a high frequency value alone is not suitable to determine tuple correctness.

### 5.3.2 Evaluation of Top25 Tuples by Pattern Spread

The tuples with largest pattern spread are depicted in table I.1. A large pattern spread is caused, if a large amount of patterns identified the same class-instance pair. The tuples with the largest pattern spread show the disadvantages of extracting only the closest noun phrases. The top-k tuples according to pattern spread mostly contain nouns, which are used frequently. This is determined by comparing the nouns with the highest pattern spread to the $25$ most frequently used nouns in the English language [39]. The result of this comparison shows, that $20$ out of the $25$ tuples contain one of these frequent nouns either as class or as instance term. This also becomes visible in the extracted tuples $\langle man, woman\rangle$ and $\langle woman, man\rangle$. In no thinkable sense, such a classification is correct. The fact the these examples contain the same nouns - once classified as instance and once classified as class - can be seen as evidence, that words which are frequently used have a high pattern spread. It is very likely that if words are used very often in the same sentence, that these are closer to the pattern than the actual entity.
A possible solution for this problem can be, to remove tuples from the database, in which both, the class term and the instance term are among the most used nouns. However, this would only partially resolve the problem and would also remove correct extractions such as $\langle point, thing\rangle$. Additionally, most of the false extractions would still be present, because in many cases only one of the entities is listed among the most frequently used nouns. On top of that there are wrong tuples, in which neither the class term nor the instance term is amongst the list of the most frequent nouns. For example the extracted tuples $\langle game, team\rangle$ or $\langle game, player\rangle$ are both wrong and do not contain a frequently used noun. Thus it becomes clear, that using lists of most frequent nouns to improve the selection will not work. The problem can only be solved, if the noun phrase extraction does not extract the closest noun phrase. A more extensive analysis on the sentence structure is required, to find out, which nouns in the sentence are actually connected by the pattern. For such a solution it is necessary to find out the dependencies of phrases in a sentence. However, such a solution will increase the computational costs of the extraction process [18].
Consequently, a large pattern spread alone does not entail tuple correctness. On the one hand it is heavily influenced by frequently used nouns. On the other hand, it is influenced by noun pairs, that are often mentioned in the same sentence. With most of these extractions being wrong, it is not suitable to take the pattern spread alone a indicator for tuple correctness.

### 5.3.3 Evaluation of Top25 Tuples by PLD Spread

A high PLD spread is caused, if a tuple was found on a large amount of different PLDs. The top-25 extractions for this attribute are depicted in table I.3. The extractions in this list are in 19 cases correct, making the PLD spread the most promising indicator for tuple correctness.
Two of the false extractions are caused by the lemmatization during the tuple normalization. The tuples $\langle parallel, worldwideleader\ in\ virtualization\rangle$ and $\langle parallel, automation\ software\rangle$ are both incorrect, because both, the software and the company, are named Parallels. The cause for this is not the extraction, but the normalization process. The company name is tagged as noun and the plural form was transformed to its lemma, in this case the singular form. These errors are difficult to solve, if the proper noun is equal to a word in a dictionary. Thus the POS tagging would need to make use of dictionaries of named entities to prevent such a result. However, such a solution cannot be recommended for a large scale extraction. For each potential named entity, a large list of names, would have to be scanned. Such a runtime increase is hardly acceptable. Another solution is to remove the lemmatization process entirely.
Another pair of false extractions are the tuples $\langle below, link\rangle$ and $\langle following, list\rangle$. These false extractions are caused by the nature of the instance terms, as both can be classified either as noun or as adverb. With a more precise POS tagging it can be expected, that the amount of falsely tagged words will decrease and in turn the attribute values of false extractions will be reduced.
Lastly, there are two false extractions in which the PLD spread has almost or exactly the same frequency. These are the tuples $\langle service, delivery\rangle$ and $\langle whoisguard\ privacy\ protection\ service, whous\ privacy\ protection\ service\rangle$. The attribute values entail, that these tuples are commonly used on a large amount of web pages with the same pattern. A plausible explanation can be found for this large pattern spread. WhoisGuard [1] is a company, that offers services for owners of a pay-level-domain, that hides personal information of of website owners. Such a re-occurring notification can be identified by a rather low pattern spread. The tuple $\langle service, delivery\rangle$ is caused by a standard phrase used by shopping websites. The reason why this tuple is found with more than one pattern is, that the class and instance terms occur frequently in the corpus. Therefore it is likely that these are also extracted with other patterns, because these frequently used terms are closest noun phrase in a handful of sentences. A solution would be to adjust the duplicate removal. If the duplicate removal would not include the PLD as uniqueness criteria of an extraction, these tuples would have significantly reduced attribute values. Such a duplicate removal would also decrease the data volume significantly, be-

[1] Company Website available at http://www.whoisguard.com/, accessed on 25.10.2015

cause large amounts of duplicate sentences can be removed.
Overall, this evaluation highlights three major improvement areas for the WTDB creation. First, the duplicate extraction has to be configured in a way, that it detects standard phrases, which are slightly altered by unique identifiers under the same PLD. Furthermore the duplicate removal has to be able to detect service notifications, which are used on a wide range of different websites. The second area of improvement is the POS tagger selection and configuration. There is a magnitude of wrongly tagged *nouns* present in the top-k lists, which not only causes wrong extractions. It also causes correct extractions to be *destroyed* during the normalization phase, because the input contained wrongly tagged words. Lastly, the concept of the closest noun phrase caused a significant amount of wrong extractions, as visible in the list of tuples, which have the largest pattern spread. As already described a more extensive analysis on the sentence structure is required to identify noun phrases that are connected by a pattern. With exception to the first improvement area, the latter ones will increase computational costs drastically during the extraction.
Furthermore, this evaluation shows that using single attribute values as indicator for tuple correctness is not sufficient. Rather a combination of values for each attribute will lead to better classifications. How to combine attribute values in a formula, that returns a probability for tuple correctness is not provided in this thesis and therefore an important part of the future work.

## 5.4 Discussion

The resulting attribute values make it obvious, that one single attribute value alone does not suffice for an automatic detection of tuple correctness. As future work a formula has to be developed for ranking tuples according to their probability to be correct. With the data present in the final resource, such a formula has to take into the account the influences on high attribute values. This includes the effects of frequently used words on the pattern spread and the influence of re-occurring standard phrases under the same PLD.
Such a ranking formula benefits from additionally generated attributes. For example frequently used words can be detected, by accumulating the frequencies or the count of each single entity. If these generated attributes are rather large for one of the entities, this can be seen as indicator that the tuple is not correct, but rather the cause often used expressions. Additionally, such a ranking formula benefits from a weighting of patterns. As the pre evaluation of patterns in Appendix B shows, that patterns differ significantly in terms of precision. These precision differences can be used to assign tuples with a better ranking, if these were detected with high

precision patterns.
Another area for further improvements is the API. At this stage it has a very basic functionality that has to be extended in the future. One requirement is the implementation of a sorting function that is not only able to sort tuples using one attribute value, but rather a combination of attribute values in form of a ranking formula. With such a feature users are able to adjust their the ranking of tuples to their preference with each single query. Additionally, the query duration can be reduced, if the API is adjusted. At this stage the API requests and returns each single piece of information stored with a tuple. However, for most use cases only a few of the attributes are required and the remaining attributes do not have to be extracted. Furthermore, there are three potentially large string values, stored for each tuple, that contain a list of patterns, PLDs and provenance IDs. Since these strings can contain up to $205.556$ entries it will increase the query duration if such a large string has to be read from the database. The solution would be to to adjust the data model of the stored tuples and introduce small samples for this strings, which only contain a small subset of the stored provenance data. During the analysis of the top-k rated tuples a further necessary feature became obvious. As some extractions, such as $\langle ability, site \rangle$, have rather high attribute values but do not make any sense for the reader, it would be nice to have a feature that returns sample sentences for each tuple. Thus the API has to introduce a new parameter for tuple queries, which indicates that samples for each query result have to be returned as well. Otherwise the user has to copy the provenance IDs of the returned tuples and start a new query to gather sentences.
As described in section 5.3.1 the duplicate sentence detection has to be improved. As already stated, these slightly modified and re-occurring sentences can be filtered using similarity measures for sentences. If the similarity is above a certain threshold, the sentence will be removed, because it is likely a re-occurring standard phrase on a web page.
The storage of sentences itself is the next point of improvement. As already mentioned each sentence is only extracted once per PLD. Therefore the stored sentences are not unique and can be aggregated without losing any information. For example the sentence that contains the tuple $\langle$ “parallel”, “worldwide leader in virtualization” $\rangle$ is stored $9375$ times. Such a duplicate removal will reduce the size of the database and in turn increase the execution speed for sentence queries.
Although the tuple extraction from the Common Crawl corpus achieved the goal of a high recall, the extraction system itself can be improved in multiple ways. First and foremost, the extracted data is written in an encoding, that does not cover the full range of characters used on the web corpus. Although the data is read in the correct encoding, some characters are replaced with question marks, when they are extracted. This error destroyed valuable information, especially for named entities

that contain characters, which are not part of the English alphabet.
The extraction itself also requires an improvement of patterns and their regex translation. During the empirical evaluation of these, a number of patterns were excluded because no robust and fast regular expression was found for these. Further work has to be done to improve the regex of the isolated and the apposition patterns. This will increase the recall even further.
Moreover, the regex for the noun phrase place-holder has to be improved, which is used for split patterns. The problem is that the place-holder will not identify abbreviations. Hence, this regex element has to include the possibility of full stops within and at the end of an entity.
Also the used patterns for extraction have to be improved in future works. Some of the used patterns have an overlap, which means that the same text passage is extracted by multiple patterns. Take for example pattern $p21a$ and $p8a$ and apply these to the following sentence: "The best fruit is an apple". This pattern overlap leads to two extracted tuples:

- $\langle apple, fruit \rangle$
- $\langle fruit, apple \rangle$

This has to be solved, for example by implementing a set of rules during the extraction phase. If a pattern overlap is detected in a line, the tuple should only be extracted once, especially if one of these patterns results in a wrong extraction.
As the careful reader might have noticed, the attribute types of the aggregated tuples have to be adjusted. At the moment the frequency is stored as *double*. The reason to select a *double* data type for the frequency was the issue regarding noun phrases, which contain multiple nouns separated by a preposition as described in section 4.1.3. The idea was split such a tuple into two tuples. Therefore the entity that contains two noun phrases is split at the preposition that separates them. Each resulting noun then becomes a tuple. However, as in most cases one of these new tuples must be wrong, the idea was to assign these tuples a frequency of $0.5$. Due to time restrictions the implementation of that idea was labelled as future work. But not only the data types require re-factoring but also the data insertion.
The data stored in the database is missing exactly five head tuples with all their modifications. The reason for that is the size limit per entry in the database. These five tuples occur so often and in so many different modifications, that the maximum allowed size of the database entry is violated. A simple way to solve this issue would be to split each tuple into multiple tuples and then insert them. This easy workaround would not harm the consistency of the WTDB, because a unique key was generated for each tuple and class and instance name combinations must not necessarily be unique. The five excluded tuple sets are displayed in table 5.7,

using their head nouns.

| Instance Head | Class Head |
|---|---|
| fig | diagram |
| fig | view |
| system | system |
| school | school |
| below | list |

Table 5.7: Tuples excluded from the WTDB

# Chapter 6

# Conclusion

The first contribution of this thesis is the development of a hyponymy relation extraction system and the resulting data in form of compressed archives (see Chapter 2 for an extensive problem introduction). The extraction system (see Chapter 4) was successful in detecting a large amount of tuples and showed that each used pattern returns correct relations. Even with filters and the removal of duplicates on archive level, the recall of the selected patterns is remarkable. The system performed well in terms of speed and robustness, because it successfully processed $98,38$ percent of a large and heterogeneous input corpus within 22 hours (see Chapter 6). Additionally the output data contains provenance information, which also allows the generation of valuable attributes.
The extracted data is then transformed to enable the aggregation of equal tuples. A system is developed that normalizes, cleans and aggregates the tuples. On the one hand, the system was successful in aggregating large amounts of data using a restricted infrastructure. On the other hand however, the normalization led to mediocre results, because it used badly POS tagged input data. The reason for badly tagged data is the combination of ill-formed input data and a extraction that prioritizes fast execution and high recall over precision. But then it is the recall, that makes it possible to generate three attributes, with remarkable value ranges. These value ranges allow a better automatic detection of tuple correctness. Especially the generated attribute PLD spread is promising as the evaluation has shown. The automatic detection requires a certain access speed to the data, which is provided with the third contribution of this thesis.
For this purpose a NoSql DBMS with inserted data is provided that stores data on disk. The selection of an on-disk database reduces the memory consumption and enables the usage of the data on a regular personal computer. The system employs a publicly available and well-known DBMS, which enables any researcher to ex-

plore the data free of charge. To enable a fast query execution using class and instance terms a redundant and nested data model is implemented, that is capable of large data output within a short duration. The main use case is to return a large amount of relations for an entity and then use the generated attributes, to determine a set of likely correct hyponymy relations. For this use case the data was duplicated and sorted according to class and instance heads. The redundant and nested data model makes it possible to execute queries without the usage of joins. The result is remarkable, as it successfully serves the main use case with up to $5.799$ returned results per second.
The API is written in one of the most popular and domain independent programming languages and therefore allows a large amount of developers to access the data with ease. First and foremost the API provides functionality to query to tuples and allows the user the specify almost every single data field stored in the WTDB. It also provides the user with a wildcard functionality for each query parameter, for a faster exploration of the resource. The API is capable of returning large amounts of data without significant memory consumption, and thus enables researchers to explore and use the data without major or probably without any investments in the existing IT infrastructure.
Overall, this work shows that it is possible to perform large scale extractions economically, in terms of duration and costs. The data is stored in a form that gives researchers the opportunity to explore one of the largest public data source for domain independent hyponymy relations without extensive investments in hardware. Furthermore the used tools and applications from third parties are licensed as GPL (General Public License) software, which enables the free usage of these tools for researchers or private users. As determined by the descriptive results in section 5.3, the extracted data is suitable to create one of the most comprehensive taxonomies to this date.

# Bibliography

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94, 2000.

[2] M. Ando, S. Sekine, and S. Ishizaki. Automatic extraction of hyponyms from japanese newspapers. using lexico-syntactic patterns. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC-2004)*, Lisbon, Portugal, May 2004. European Language Resources Association (ELRA).

[3] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[5] C. Castella Xavier, V.L. Strube de Lima, and M. Souza. Open information extraction based on lexical-syntactic patterns. In *Intelligent Systems (BRACIS), 2013 Brazilian Conference on*, pages 189–194, Oct 2013.

[6] R. Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.

[7] Marion G Ceruti. An expanded review of information-system terminology. Technical report, DTIC Document, 1999.

[8] J. Cho. *Crawling the Web: Discovery and maintenance of large-scale Web data*. PhD thesis, Stanford University, 2001.

[9] J. Christensen, S. Soderland, and O. Etzioni. An Analysis of Open Information Extraction Based on Semantic Role Labeling Categories and Subject Descriptors. *Proceeding of K-CAP '11 Proceedings of the sixth international conference on Knowledge capture*, pages 113–119, 2011.

[10] D. Crystal. *A Dictionary of Linguistics and Phonetics*. Blackwell, Malden, 6. edition, 2008.

[11] Klaas Dellschaft and Steffen Staab. Measuring the similarity of concept hierarchies and its influence on the evaluation of learning procedures. *Master's Thesis (Diplomarbeit), University of Koblenz-Landau*, 2005.

[12] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA, 2014. ACM.

[13] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.

[14] A. Fader, S. Soderland, and O. 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] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

[16] Common Crawl Foundation. Common crawl - about. Accessed on 25.10.2015.

[17] Common Crawl Foundation. Common crawl - so you're ready to get started. Accessed on 25.10.2015.

[18] P. Gamallo, M. Garcia, and S. Fernández-Lanza. Dependency-based open information extraction. *EACL 2012*, page 10, 2012.

[19] Pierdaniele Giaretta. Ontologies and knowledge bases towards a terminological clarification. *Towards very large knowledge bases: knowledge building & knowledge sharing*, 25:32, 1995.

[20] R. Gupta, A. Halevy, X. Wang, S. Whang, and F. Wu. Biperpedia: An ontology for search applications. *Proc. VLDB Endow.*, 7(7):505–516, March 2014.

[21] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[22] MongoDB Inc. Mongodb limits and thresholds. Accessed on 25.10.2015.

[23] H. Jing, E. Haihong, L. Guan, and D. Jian. Survey on nosql database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366, Oct 2011.

[24] P. D. Karp. What database management system (s) should be employed in bioinformatics applications? *OMICS A Journal of Integrative Biology*, 7(1):35–36, 2003.

[25] C. Klaussner and D. Zhekova. Pattern-based ontology construction from selected wikipedia pages. In *Proceedings of the Student Research Workshop associated with RANLP 2011*, pages 103–108, 2011.

[26] T. Lin, Mausam, and O. Etzioni. Identifying functional relations in web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1266–1276, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[27] P. Mell and T. Grance. The nist definition of cloud computing. 2011.

[28] Y. Merhav. A weighting scheme for open information extraction. *NAACL-HLT 2012*, page 60, 2012.

[29] Robert Meusel, Hannes Mühleisen, Oliver Lehmberg, Petar Petrovski, and Christian Bizer. Web data commons - extraction framework. Accessed on 25.10.2015.

[30] C. Orna-Montesinos. Words and patterns: lexico-grammatical patterns and semantic relations in domain-specific discourses. 2011.

[31] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64. ACM, 2008.

[32] D. C. Pastoors. Web-scale knowledge harvesting: Populating the webisa taxonomy using the web, 2014.

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

[34] P.R. Ray, V. Harish, S. Sarkar, and A. Basu. Part of speech tagging and local word grouping techniques for natural language parsing in hindi. In *Proceedings of ICON 2003*, 2003.

[35] A. Ritter and O. Etzioni. A latent dirichlet allocation method for selectional preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 424–434. Association for Computational Linguistics, 2010.

[36] A. Ritter, S. Soderland, and O. Etzioni. What is this, anyway: Automatic hypernym discovery. 2009.

[37] B. N. Rossiter. Introduction to data base management systems. *Computer Physics Communications*, 33:5 – 12, 1984.

[38] T. Stubblebine. *Regular expression pocket reference : "Regular expressions for Perl, Ruby, PHP, Python, C, Java, and .NET".* Safari Books Online. O'Reilly, Sebastopol, Calif., 2nd edition, 2007.

[39] Oxford university press. The oec: Facts about the language. Accessed on 25.10.2015.

[40] M. Ushold and R. Jasper. A framework for understanding and classifying ontology application. In *Proc IJCAI99 Workshop on Ontologies and Problem-Solving Methods. Stockholm*, volume 282, 1999.

[41] J. Varia. Cloud architectures. page 16, 2008.

[42] Feng-Kwei Wang. Designing a case-based e-learning system: what, how and why. *Journal of Workplace Learning*, 14(1):30–43, 2002.

[43] D. S. Weld, R. Hoffmann, and F. Wu. Using wikipedia to bootstrap open information extraction. *ACM SIGMOD Record*, 37(4):62, 2009.

[44] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu. Cloud storage as the infrastructure of cloud computing. In *Intelligent Computing and Cognitive Informatics (ICICCI), 2010*, pages 380–383. IEEE, 2010.

[45] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 481–492, New York, USA, 2012. ACM.

[46] A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. 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*, NAACL-Demonstrations '07, pages 25–26, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[47] T. Yigit, M.A. Cakar, and A.S. Yuksel. The experience of nosql database in telecommunication enterprise. In *Application of Information and Communication Technologies (AICT), 2013 7th International Conference on*, pages 1–4, Oct 2013.

# Appendix A

# Program Code / Resources

The documented source code, an API description, third party software, a PDF version of this thesis is contained on a CD attached to this thesis. The Files are listed in the following table:

| Name | Description |
|---|---|
| cleaning-app | Contains source code and an executable jar-file for Duplicate Removal, Data Normalization and Data Aggregation |
| insert-app | Contains source code and an executable jar-file for inserting unique tuples into a MongoDB database |
| query-app | Contains source code and an executable jar-file for the API to query the final resource |
| Description_API.pdf | Contains a descriptions for inserting the data and querying the data using |
| Patterns.pdf | Contains a list of used patterns for hyponymy extraction |
| extraction-framework | Contains the source code for the hyponymy extraction from the Common Crawl corpus. |
| Third-Party-Tools | Third Party Tools used in this work. |
| thesis-jseitner.pdf | This master thesis in .pdf format. |

# Appendix B

# Patterns

| ID | Description | Tuples found | Type | Reference |
|---|---|---|---|---|
| p1 | $NP_I$ and other $NP_C$ | 45900092 | compact | [21] |
| p2 | $NP_C$ especially $NP_I$ | 20872227 | compact | [21] |
| p3a | $NP_C$ including $NP_I$ | 80640885 | compact | [21] |
| p4 | $NP_I$ or other $NP_C$ | 13392348 | compact | [21] |
| p5 | $NP_C$ such as $NP_I$ | 70337543 | compact | [21] |
| p6 | $NP_I$ and any other $NP_C$ | 975735 | compact | [32] |
| p7 | $NP_I$ and some other $NP_C$ | 296524 | compact | [32] |
| p8 | $NP_I$ is a $NP_C$ | 187644160 | compact | [33] |
| p8b | $NP_I$ was a $NP_C$ | 39585428 | compact | [33] |
| p8c | $NP_I$ are a $NP_C$ | 15141131 | compact | [33] |
| p8d | $NP_I$ were a $NP_C$ | 3206238 | compact | [33] |
| p9 | $NP_C$ like $NP_I$ | n.a. | compact | [33] |
| p10 | Such $NP_C$ as $NP_I$ | 5755389 | split | [21] |
| p11 | $NP_I$ like other $NP_C$ | 402388 | compact | [33] |
| p12a | $NP_I$, one of the $NP_C$ | 4200376 | compact | [33] |
| p12b | $NP_I$, one of these $NP_C$ | 53235 | compact | [33] |
| p12c | $NP_I$, one of those $NP_C$ | 99241 | compact | [33] |
| p13 | Examples of $NP_C$ is $NP_I$ | 267021 | split | [30] |
| p14 | Examples of $NP_C$ are $NP_I$ | 267764 | split | [30] |
| p15a | $NP_I$ are examples of $NP_C$ | 2205089 | compact | [30] |
| p15b | $NP_I$ is example of $NP_C$ | 292706 | compact | [30] |
| p16 | $NP_C$ for example $NP_I$ | 2356522 | compact | [30] |
| p20a | $NP_I$ is adj. sup. $NP_C$ | 6150245 | compact | [30] |
| p20b | $NP_I$ are adj. sup. $NP_C$ | 1393484 | compact | [30] |
| p20c | $NP_I$ is adj. sup. most $NP_C$ | 2286478 | compact | [30] |
| p20d | $NP_I$ are adj. sup. most $NP_C$ | 860770 | compact | [30] |

| ID | Description | Tuples found | Type | Source |
|---|---|---|---|---|
| p21a | Adj. sup. $NP_C$ is $NP_I$ | 10360953 | split | [30] |
| p21b | Adj. sup. $NP_C$ are $NP_I$ | 3755893 | split | [30] |
| p21c | Adj. sup. $NP_C$ is $NP_I$ | 2999877 | split | [30] |
| p21d | Adj. sup. most $NP_C$ are $NP_I$ | 2357968 | split | [30] |
| p22a | $NP_I$ which is called $NP_C$ | 119317 | compact | [2] |
| p22b | $NP_I$ which is named $NP_C$ | 19122 | compact | [2] |
| p23a | $NP_C$ mainly $NP_I$ | 4792792 | compact | [2] |
| p23b | $NP_C$ mostly $NP_I$ | 8383063 | compact | [2] |
| p23c | $NP_C$ notably $NP_I$ | 1154745 | compact | [2] |
| p23d | $NP_C$ particularly $NP_I$ | 11656254 | compact | [2] |
| p23e | $NP_C$ principally $NP_I$ | 455578 | compact | [2] |
| p24 | $NP_C$ in particular $NP_I$ | 2354596 | compact | [25] |
| p25 | $NP_C$ except $NP_I$ | 9648662 | compact | [25] |
| p26 | $NP_C$ other than $NP_I$ | 7175087 | compact | [25] |
| p27a | $NP_C$ e.g. $NP_I$ | 1973022 | compact | [25] |
| p27b | $NP_C$ i.e. $NP_I$ | 2114793 | compact | [25] |
| p28a | $NP_I$, a kind of $NP_C$ | 1452822 | compact | [25] |
| p28b | $NP_I$, kinds of $NP_C$ | 4618873 | compact | [25] |
| p28c | $NP_I$, a form of $NP_C$ | 1127173 | compact | [25] |
| p28d | $NP_I$, forms of $NP_C$ | 3326957 | compact | [25] |
| p29a | $NP_I$ which look like $NP_C$ | 68945 | compact | [2] |
| p29c | $NP_I$ which sound like $NP_C$ | 32730 | compact | [2] |
| p30a | $NP_C$ which are similar to $NP_I$ | 17304 | compact | [25] |
| p30b | $NP_C$ which is similar to $NP_I$ | 63713 | compact | [25] |
| p31a | $NP_C$ example of this is $NP_I$ | 14237 | compact | [30] |
| p31b | $NP_C$ examples of this are $NP_I$ | 1515 | compact | [30] |
| p32 | $NP_C$, $NP_I$ for example | n.a. | isolated | [30] |
| p34 | $NP_C$ types $NP_I$ | 11080276 | compact | [30] |
| p35 | $NP_I$, $NP_C$ types | n.a. | isolated | [30] |
| p36 | $NP_C$ whether $NP_I$ or | 2800349 | split | [30] |
| p37 | Compare $NP_I$ with $NP_C$ | 340636 | split | [30] |
| p38 | $NP_C$ compared to $NP_I$ | 346525 | compact | [30] |
| p39 | $NP_C$ among them $NP_I$ | 524784 | compact | [30] |
| p40 | $NP_C$ as $NP_I$ | n.a. | compact | [30] |
| p42 | $NP_I$ or the many $NP_C$ | 15192 | compact | [30] |
| p43 | $NP_I$ sort of $NP_C$ | 7884398 | compact | [30] |

# Appendix C

# Regex of Compact patterns

**p1:** $NP_I$ **and other** $NP_C$

```
(\p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sand \sother \s [ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]? ( \p{L}| \d)
```

**p2:** $NP_C$ **especially** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sespecially \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p3a:** $NP_C$ **including** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sincluding \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p4:** $NP_I$ **or other** $NP_C$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sor \sother \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p5:** $NP_C$ **such as** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \ssuch \sas \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p6:** $NP_I$ **and any other** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sand \sany \sother \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p7:** $NP_I$ **and some other** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sand \ssome \sother \s[ \u0027 \u2018
\u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p8a:** $NP_I$ **is a** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sis \sa \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p8b:** $NP_I$ **was a** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \swas \sa \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p8c:** $NP_I$ **are a** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sare \sa \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p8d:** $NP_I$ **were a** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \swere \sa \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p9:** $NP_C$ **like** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \slike \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p11:** $NP_I$ **like other** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \slike \sother \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p12a:** $NP_I$**, one of the** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \, \sone \sof \sthe \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p12b:** $NP_I$**, one of these** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \, \sone \sof \sthese \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p12c:** $NP_I$**, one of those** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \, \sone \sof \sthose \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p15a:** $NP_I$ **are examples of** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sexamples \sof \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p15b:** $NP_I$ **is example of** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sis \san \sexample \sof \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p16:** $NP_C$ **for example** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \sfor \sexample \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p20a:** $NP_I$ **is adj. sup.** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sis \sthe \s \w+est \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p20b:** $NP_I$ **are adj. sup.** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sare \sthe \s \w+est \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p20c:** $NP_I$ **is adj. sup. most** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sis \sthe \smost \s \w+ \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p20d:** $NP_I$ **are adj. sup. most** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sare \sthe \smost \s \w+ \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p22a:** $NP_I$ **which is called** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhich \sis \scalled \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p22b:** $NP_I$ **which is named** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhich \sis \snamed \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p23a:** $NP_C$ **mainly** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \smainly \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p23b:** $NP_C$ **mostly** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \smostly \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p23c:** $NP_C$ **notably** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \snotably \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p23d:** $NP_C$ **particularly** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \sparticularly \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p23e:** $NP_C$ **principally** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sprincipally \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p24:** $NP_C$ **in particular** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sin \sparticular \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p25:** $NP_C$ **except** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sexcept \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p26:** $NP_C$ **other than** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \sother \sthan \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p27a:** $NP_C$ **e.g.** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \se \.g \. \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p27b:** $NP_C$ **i.e.** $NP_I$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \,? \si \.e \. \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p28a:** $NP_I$**, a kind of** $NP_C$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \, \sa \skind \sof \s[ \u0027 \u2018 \u2019
\u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p28b:** $NP_I$**, kinds of** $NP_C$

```
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u00A9 \u00AE]? \, \skinds \sof \s[ \u0027 \u2018 \u2019 \u201A
\u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)
```

**p28c:** $NP_I$**, a form of** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \, \sa \sform \sof \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p28d:** $NP_I$**, forms of** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \, \sforms \sof \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p29a:** $NP_I$ **which look like** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhich \slooks? \slike \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p29c:** $NP_I$ **which sound like** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhich \ssounds? \slike \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p30a:** $NP_C$ **which are similar to** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhich \sare \ssimilar \sto \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p30b:** $NP_C$ **which is similar to** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhich \sis \ssimilar \sto \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p31a:** $NP_C$ **example of this is** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \sexample \sof \sthis \sis \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p31b:** $NP_C$ **examples of this are** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \sexamples \sof \sthis \sare \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p34:** $NP_C$ **types** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \stypes \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p38:** $NP_C$ **compared to** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \scompared \sto \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p39:** $NP_C$ **among them** $NP_I$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \samong \sthem \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p40:** $NP_C$ **as** $NP_I$
( \p \L \| \d)[\u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \sas \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p \L \| \d)

**p42:** $NP_I$ **or the many** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \sor \sthe \smany \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p43:** $NP_I$ **sort of** $NP_C$
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \ssort \sof \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

# Appendix D

# Regex of Split Patterns

**p10: Such $NP_C$ as $NP_I$**

```
(?>(S|s)uch \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E
\u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012
\u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018
\u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]?
\s){1,4}as \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E
\u201F \u0022]?( \p{L}| \d))
```

**p13: Examples of $NP_C$ is $NP_I$**

```
(?>(E|e)xample \sof \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011
\u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027
\u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]?
\s){1,4}is \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E
\u201F \u0022]?( \p{L}| \d))
```

**p14: Examples of $NP_C$ are $NP_I$**

```
(?>(E|e)xamples \sof \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D
\u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011
\u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027
\u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]?
\s){1,4}are \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E
\u201F \u0022]?( \p{L}| \d))
```

**p21a: Adj. sup.** $NP_C$ **is** $NP_I$
(?> \p{L}+est \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \s){1,4}is \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d))

**p21b: Adj. sup.** $NP_C$ **are** $NP_I$
(?> \p{L}+est \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \s){1,4}are \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d))

**Adj. sup. most** $NP_C$ **is** $NP_I$
(?>(M|m)ost \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \s){2,5}is \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d))

**Adj. sup. most** $NP_C$ **are** $NP_I$
(?>(M|m)ost \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \s){2,5}are \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d))

**p36:** $NP_C$ **whether** $NP_I$ **or**
( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \,? \swhether \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \s){1,4}or \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d)

**p37: Compare $NP_I$ with $NP_C$**
(?>(C|c)ompare \s([ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D \u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u00A9 \u00AE]? \s){1,4}with \s[ \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F \u0022]?( \p{L}| \d))

# Appendix E

# Regex of Isolated Patterns

**p32:** $NP_C$, $NP_I$ **for example**

```
(?>( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C
\u201D \u201E \u201F \u00A9 \u00AE]?([ \u0027 \u2018 \u2019 \u201A \u201B
\u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D
\u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022
\u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F
\u00A9 \u00AE]? \,? \s){1,4}for \sexample( \s| \.| \?| \!| \,))
```

**p35:** $NP_I$, $NP_C$ **types**

```
(?>( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C
\u201D \u201E \u201F \u00A9 \u00AE]?([ \u0027 \u2018 \u2019 \u201A \u201B
\u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D
\u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022
\u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F
\u00A9 \u00AE]? \s){1,4}types?( \s| \.| \?| \!| \,))
```

**p41:** $NP_C$**,** $NP_I$ **for instance**

```
(?>( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C
\u201D \u201E \u201F \u00A9 \u00AE]?([ \u0027 \u2018 \u2019 \u201A \u201B
\u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D
\u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022
\u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F
\u00A9 \u00AE]? \,? \s){1,4}for \sinstance( \s| \.| \?| \!| \,))
```

**p44:** $NP_C$**,** $NP_I$ **and the like**

```
(?>( \p{L}| \d)[ \u0022 \u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C
\u201D \u201E \u201F \u00A9 \u00AE]?([ \u0027 \u2018 \u2019 \u201A \u201B
\u201C \u201D \u201E \u201F \u0022]?( \p{L}++| \d++ \p{L}++)([ \u002D
\u2010 \u2011 \u2012 \u2013 \u2014 \u2015 \u2043]?( \p{L}++| \d++))?[ \u0022
\u0026 \u0027 \u2018 \u2019 \u201A \u201B \u201C \u201D \u201E \u201F
\u00A9 \u00AE]? \s){1,4}and \sthe \slike( \s| \.| \?| \!| \,))
```

## Appendix F

# Abbreviations for Sentence Splitter

The following abbreviations are provided by the European Parliament Proceedings Parallel Corpus 1996-2011 [1], with exception to the ones, which are denoted with an asterisk *. These are added by the author due to their observed frequency in the web corpus.

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, Adj, Adm, Adv, Art*, Asst, Bart, Bldg, Brig, Bros, Capt, Cmdr, Col, Comdr, Con, Corp, Cpl, DR, Dr, Drs, Ens, Fig*, Gen, Gov, Hon, Hr, Hosp, Insp, Lt, MM, MR, MRS, MS, Maj, Messrs, Mlle, Mme, Mr, Mrs, Ms, Msgr, No*, Nos*, Nr*, Op, Ord, Pat, Pfc, Ph, pp*, Prof, Pvt, Rep, Reps, Res, Rev, Rt, Sen, Sens, Sfc, Sgt, Sr, St, Supt, Surg, v, vs, U.S*, U.K*, i.e, rev, e.g,

[1]Downloaded from http://www.statmt.org/europarl/v7/tools.tgz on 7.10.2015

Appendix G

# Empirically Determined Collective Nouns

- array
- couple
- collection
- group
- number
- range
- selection
- series
- set
- variety

# Appendix H

# Evaluation of patterns

The following instructions to evaluate the correctness of an extracted hyponymy tuple are copied from Ponzetto and Strube [33]:

Below is a list of pairs of words. For each pair (a, b), please assign one of the following relations: IOF a is an INSTANCE-OF b corresponds to set membership. a must refer to a (unique) individual and b must refer to a set such that a is a member of b. Examples:

North Korea IOF country
Errol Morris IOF film director

ISA a ISA b corresponds to set inclusion. a and b must refer sets such that a is a (proper) subset of b. Examples:

physicists ISA scientists
football ISA sport

NOT If none of the above apply.

When annotating the pairs, please follow these guidelines:
1. Annotate the pairs (a, b) while answering the question: is a a (kind of/form of) b?
2. Reify the concepts, that is, consider abstract concepts as made up of material objects. E.g. disciplines are made up of things like publications, concrete theories, therefore:

psychology ISA social science
natural language processing ISA artificial intelligence

3. Disregard the number of the phrases (i.e. singular or plural), e.g. theoretical physicists ISA scientist, although scientist is singular and would denote a single individual rather than a set;

4. If a phrase has multiple senses, consider all senses of the phrase, e.g. school can refer to both the building and the institution so these pairs would be tagged as follows:

   school ISA building
   school ISA institution

In other words, tag with a relation if there is *at least one* pair of senses of the two phrases which is in the relation. Note therefore that the same phrase can be in an ISA relation with two distinct sets (i.e. sets whose intersection is empty, as in the example above).

| ID | Description | Hits | ISA | IOF | NOT | Checked | Ratio |
|---|---|---|---|---|---|---|---|
| p1 | $NP_I$ and other $NP_C$ | 2469 | 60 | 10 | 30 | 100 | 0,7 |
| p2 | $NP_C$ especially $NP_I$ | 1655 | 15 | 4 | 81 | 100 | 0,19 |
| p3a | $NP_C$ including $NP_I$ | 3936 | 32 | 12 | 56 | 100 | 0,44 |
| p4 | $NP_I$ or other $NP_C$ | 888 | 69 | 1 | 30 | 100 | 0,7 |
| p5 | $NP_C$ such as $NP_I$ | 2862 | 52 | 6 | 42 | 100 | 0,58 |
| p6 | $NP_I$ and any other $NP_C$ | 70 | 74 | 2 | 24 | 100 | 0,76 |
| p7 | $NP_I$ and some other $NP_C$ | 17 | 48 | 6 | 46 | 100 | 0,54 |
| p8a | $NP_I$ is a $NP_C$ | 15365 | 29 | 15 | 56 | 100 | 0,44 |
| p8b | $NP_I$ was a $NP_C$ | 4214 | 18 | 21 | 61 | 100 | 0,39 |
| p8c | $NP_I$ are a $NP_C$ | 1002 | 52 | 5 | 43 | 100 | 0,57 |
| p8d | $NP_I$ were a $NP_C$ | 171 | 36 | 6 | 58 | 100 | 0,42 |
| p9 | $NP_C$ like $NP_I$ | 20240 | 14 | 3 | 83 | 100 | 0,17 |
| p10 | Such $NP_C$ as $NP_I$ | 380 | 54 | 4 | 42 | 100 | 0,58 |
| p11 | $NP_I$ like other $NP_C$ | 31 | 20 | 11 | 69 | 100 | 0,31 |
| p12a | $NP_I$, one of the $NP_C$ | 240 | 17 | 21 | 62 | 100 | 0,38 |
| p12b | $NP_I$, one of these $NP_C$ | 4 | 12 | 1 | 87 | 100 | 0,13 |
| p12c | $NP_I$, one of those $NP_C$ | 8 | 10 | 5 | 85 | 100 | 0,15 |
| p13 | Examples of $NP_C$ is $NP_I$ | 32 | 33 | 0 | 67 | 100 | 0,33 |
| p14 | Examples of $NP_C$ are $NP_I$ | 15 | 41 | 4 | 55 | 100 | 0,45 |
| p15a | $NP_I$ are examples of $NP_C$ | 192 | 16 | 4 | 80 | 100 | 0,2 |
| p15b | $NP_I$ is example of $NP_C$ | 34 | 30 | 6 | 64 | 100 | 0,36 |
| p16 | $NP_C$ for example $NP_I$ | 128 | 30 | 1 | 69 | 100 | 0,31 |
| p20a | $NP_I$ is adj. sup. $NP_C$ | 793 | 49 | 14 | 37 | 100 | 0,63 |
| p20b | $NP_I$ are adj. sup. $NP_C$ | 176 | 39 | 2 | 59 | 100 | 0,41 |
| p20c | $NP_I$ is adj. sup. most $NP_C$ | 254 | 52 | 11 | 37 | 100 | 0,63 |
| p20c | $NP_I$ is adj. sup. most $NP_C$ | 87 | 46 | 3 | 51 | 100 | 0,49 |
| p21a | Adj. sup. $NP_C$ is $NP_I$ | 960 | 24 | 1 | 75 | 100 | 0,25 |
| p21b | Adj. sup. $NP_C$ are $NP_I$ | 334 | 18 | 1 | 81 | 100 | 0,19 |
| p21c | Adj. sup. $NP_C$ is $NP_I$ | 402 | 28 | 3 | 69 | 100 | 0,31 |
| p21d | Adj. sup. most $NP_C$ are $NP_I$ | 369 | 19 | 2 | 79 | 100 | 0,21 |

Table H.1: Precision and Recall Measurement for Patterns (Part 1)

| Desc. | ID | Hits | ISA | IOF | NOT | Checked | Ratio |
|---|---|---|---|---|---|---|---|
| p22a | $NP_I$ which is called $NP_C$ | 8 | 39 | 11 | 50 | 100 | 0,5 |
| p22b | $NP_I$ which is named $NP_C$ | 1 | 21 | 5 | 74 | 100 | 0,26 |
| p23a | $NP_C$ mainly $NP_I$ | 285 | 22 | 0 | 78 | 100 | 0,22 |
| p23b | $NP_C$ mostly $NP_I$ | 643 | 16 | 0 | 84 | 100 | 0,16 |
| p23c | $NP_C$ notably $NP_I$ | 55 | 20 | 8 | 72 | 100 | 0,28 |
| p23d | $NP_C$ particularly $NP_I$ | 764 | 18 | 1 | 81 | 100 | 0,19 |
| p23e | $NP_C$ principally $NP_I$ | 23 | 24 | 2 | 74 | 100 | 0,26 |
| p24 | $NP_C$ in particular $NP_I$ | 118 | 24 | 1 | 75 | 100 | 0,25 |
| p25 | $NP_C$ except $NP_I$ | 872 | 20 | 2 | 78 | 100 | 0,22 |
| p26 | $NP_C$ other than $NP_I$ | 541 | 39 | 5 | 56 | 100 | 0,44 |
| p27a | $NP_C$ e.g. $NP_I$ | 127 | 32 | 1 | 67 | 100 | 0,33 |
| p27b | $NP_C$ i.e. $NP_I$ | 123 | 28 | 1 | 71 | 100 | 0,29 |
| p28a | $NP_I$, a kind of $NP_C$ | 6 | 23 | 1 | 76 | 100 | 0,24 |
| p28b | $NP_I$, kinds of $NP_C$ | 0 | 16 | 1 | 79 | 96 | 0,18 |
| p28c | $NP_I$, a form of $NP_C$ | 4 | 45 | 0 | 55 | 100 | 0,45 |
| p28d | $NP_I$, forms of $NP_C$ | 1 | 18 | 0 | 82 | 100 | 0,18 |
| p29a | $NP_I$ which look like $NP_C$ | 6 | 33 | 0 | 67 | 100 | 0,33 |
| p29c | $NP_I$ which sound like $NP_C$ | 2 | 13 | 0 | 87 | 100 | 0,13 |
| p30a | $NP_C$ which are similar to $NP_I$ | 2 | 18 | 0 | 82 | 100 | 0,18 |
| p30b | $NP_C$ which is similar to $NP_I$ | 4 | 28 | 0 | 72 | 100 | 0,28 |
| p31a | $NP_C$ example of this is $NP_I$ | 6 | 28 | 1 | 71 | 100 | 0,29 |
| p31b | $NP_C$ examples of this are $NP_I$ | 0 | 4 | 0 | 12 | 16 | 0,25 |
| p32 | $NP_C$ $NP_I$ for example | 567 | 17 | 1 | 82 | 100 | 0,18 |
| p34 | $NP_C$ types $NP_I$ | 944 | 17 | 0 | 83 | 100 | 0,17 |
| p35 | $NP_I$ $NP_C$ types | 3416 | 11 | 1 | 88 | 100 | 0,12 |
| p36 | $NP_C$ whether $NP_I$ or | 297 | 12 | 1 | 87 | 100 | 0,13 |
| p37 | Compare $NP_I$ with $NP_C$ | 41 | 16 | 0 | 84 | 100 | 0,16 |
| p38 | $NP_C$ compared to $NP_I$ | 450 | 13 | 4 | 83 | 100 | 0,17 |
| p39 | $NP_C$ among them $NP_I$ | 26 | 11 | 12 | 77 | 100 | 0,23 |
| p40 | $NP_I$ as $NP_C$ | 44015 | 16 | 1 | 83 | 100 | 0,17 |
| p41 | $NP_C$ $NP_I$ for instance | 128 | 13 | 0 | 87 | 100 | 0,13 |
| p42 | $NP_I$ or the many $NP_C$ | 1 | 26 | 5 | 69 | 100 | 0,31 |
| p43 | $NP_I$ sort of $NP_C$ | 770 | 16 | 2 | 82 | 100 | 0,18 |
| p44 | $NP_C$ $NP_I$ and the like | 47 | 14 | 0 | 86 | 100 | 0,14 |

Table H.2: Precision and Recall Measurement for Patterns (Part 2)

# Appendix I

# Top Tuples Per Attribute

## I.1 Top 25 - Pattern Spread

| No. | Instance | Class | Freq. | Patterns | PLDs |
|---|---|---|---|---|---|
| 1 | woman | man | 3797 | 47 | 1924 |
| 2 | time | thing | 5655 | 47 | 2900 |
| 3 | man | woman | 3925 | 46 | 1999 |
| 4 | man | thing | 2409 | 46 | 1245 |
| 5 | book | thing | 3806 | 46 | 1894 |
| 6 | parent | child | 2059 | 46 | 1329 |
| 7 | china | country | 25726 | 46 | 7877 |
| 8 | people | thing | 7881 | 45 | 3830 |
| 9 | something | thing | 2723 | 45 | 1540 |
| 10 | time | people | 2161 | 45 | 1396 |
| 11 | child | family | 6488 | 45 | 3524 |
| 12 | child | people | 28541 | 45 | 7706 |
| 13 | country | people | 1039 | 45 | 696 |
| 14 | english | language | 22551 | 45 | 8801 |
| 15 | time | game | 947 | 44 | 564 |
| 16 | player | game | 1041 | 44 | 611 |
| 17 | life | thing | 5481 | 44 | 2023 |
| 18 | idea | thing | 1542 | 44 | 1006 |
| 19 | point | thing | 1190 | 44 | 768 |
| 20 | school | child | 1377 | 44 | 915 |
| 21 | game | player | 1130 | 44 | 638 |
| 22 | person | people | 1257 | 44 | 849 |
| 23 | game | team | 1050 | 43 | 567 |
| 24 | god | thing | 4597 | 43 | 1645 |
| 25 | mind | thing | 1722 | 43 | 966 |

Table I.1: Top 25 Tuples by Pattern Spread

## I.2 Top 25 - Frequency

| No. | Instance | Class | Freq. | Patterns | PLDs |
|---|---|---|---|---|---|
| 1 | worldcat | linked data resources | 205556 | 1 | 2 |
| 2 | viaf | linked data resources | 204505 | 1 | 2 |
| 3 | active member of questia | rest | 143633 | 1 | 1 |
| 4 | march 12 | employment for pay period | 106354 | 1 | 3 |
| 5 | below | map | 105701 | 9 | 795 |
| 6 | conditionscontact usbuilt by orange | cooperative of great diversity | 81370 | 1 | 1 |
| 7 | rights reservedabout magnumterm | cooperative of great diversity | 81329 | 1 | 1 |
| 8 | href= target='_top'>more database select | datum collection | 81180 | 1 | 1 |
| 9 | lowest load time | highest load time | 76567 | 1 | 3 |
| 10 | conditionscontact usbuilt by orange | distinction | 69010 | 1 | 1 |
| 11 | rights reservedabout magnumterm | distinction | 68969 | 1 | 1 |
| 12 | share page | website by copying | 63884 | 1 | 1 |
| 13 | worldcat | linked data resource | 63874 | 1 | 1 |
| 14 | share page | pasting link http | 63769 | 1 | 1 |
| 15 | viaf | linked data resource | 63563 | 1 | 1 |
| 16 | post | website by copying | 60872 | 1 | 1 |
| 17 | post | pasting link http | 60774 | 1 | 1 |
| 18 | ability | site | 53184 | 16 | 166 |
| 19 | send there | error | 51610 | 1 | 2 |
| 20 | year of archive content | com | 47978 | 1 | 4 |
| 21 | member learn | hal leonard digital item | 47391 | 1 | 1 |
| 22 | address | basic information | 43129 | 6 | 151 |
| 23 | title | basic information | 42999 | 4 | 33 |
| 24 | telephone number | basic information | 42979 | 3 | 21 |
| 25 | executive name | basic information | 42959 | 1 | 1 |

Table I.2: Top 25 Tuples by Frequency.

## I.3 Top 25 - Pay-Level-Domain Spread

| No. | Instance | Class | Freq. | Patterns | PLDs |
|---|---|---|---|---|---|
| 1 | name | information | 16186 | 30 | 10096 |
| 2 | parallel | worldwide leader in virtualization | 9377 | 1 | 9375 |
| 3 | parallel | automation software | 9371 | 1 | 9366 |
| 4 | service | delivery | 9352 | 11 | 9348 |
| 5 | english | language | 22551 | 45 | 8801 |
| 6 | united states | country | 24850 | 42 | 8596 |
| 7 | name | personal information | 10112 | 18 | 8213 |
| 8 | following | list | 16229 | 13 | 8043 |
| 9 | china | country | 25726 | 46 | 7877 |
| 10 | whoisguard privacy protection service | whous privacy protection service | 7743 | 1 | 7743 |
| 11 | child | people | 28541 | 45 | 7706 |
| 12 | year | long time | 19043 | 22 | 7313 |
| 13 | google | search engine | 14437 | 39 | 6853 |
| 14 | patience | virtue | 13917 | 18 | 6173 |
| 15 | canada | country | 12357 | 43 | 6140 |
| 16 | india | country | 15989 | 43 | 6127 |
| 17 | today | day | 15013 | 38 | 6044 |
| 18 | free lunch | thing | 13963 | 5 | 5905 |
| 19 | cancer | disease | 16295 | 39 | 5888 |
| 20 | facebook | social media | 11318 | 39 | 5821 |
| 21 | below | link | 12574 | 11 | 5658 |
| 22 | imitation | form of flattery | 11803 | 6 | 5519 |
| 23 | twitter | social media | 10303 | 36 | 5138 |
| 24 | today | good day | 12906 | 7 | 4963 |
| 25 | japan | country | 10641 | 38 | 4912 |

Table I.3: Top 25 Tuples by Pay-Level-Domain Spread.

# Appendix J

# Query Sets

The following pages depict the queries used to test the duration of queries are presented.
First, the queries which have both, class and instance term specified, are presented. These tables contain the exact class and instance terms, as well as the query duration. The amount of returned results are not depicted in these tables, because the class instance pairs are per definition unique and only return one result.
Second, the queries are presented, in which only the class term is specified.
Lastly, the queries for tuples are depicted, in which only the instance terms are specified. For these query sets, also the amount of returned results is listed.

| Instance | Class | Duration(ms) |
|---|---|---|
| name | information | 580 |
| below | list | 10828 |
| name | information | 24980 |
| parallel | worldwide leader in virtualization | 7780 |
| parallel | automation software | 11489 |
| service | delivery | 19430 |
| english | language | 6926 |
| united states | country | 49483 |
| name | personal information | 1304 |
| following | list | 853 |
| china | country | 2622 |
| whoisguard privacy protection service | whous privacy protection service | 30605 |
| child | people | 12537 |
| year | long time | 6985 |
| google | search engine | 7941 |
| patience | virtue | 7160 |
| canada | country | 2660 |
| india | country | 2327 |
| today | day | 8340 |
| free lunch | thing | 17283 |
| cancer | disease | 20475 |
| facebook | social media | 27144 |
| below | link | 20233 |
| imitation | form of flattery | 15490 |
| twitter | social media | 785 |
| today | good day | 2478 |
| japan | country | 2826 |
| us | country | 2508 |
| answer | resounding yes | 3065 |
| germany | country | 2585 |
| alcohol | drug | 4943 |
| below | example | 14556 |
| address | information | 27709 |
| australia | country | 2896 |
| facebook | social network | 7525 |
| trademark | proprietary rights | 5015 |
| parent | family member | 1867 |
| laughter | medicine | 874 |
| dog | animal | 28016 |
| information | material | 30674 |
| virus | harmful component | 3012 |
| reasonable attorney fee | expense | 8407 |
| com | website | 7109 |
| bank | financial institution | 30284 |
| people | thing | 3160 |
| there | time | 7215 |
| life | journey | 4591 |
| product | material | 8697 |
| honesty | policy | 21030 |
| woman | people | 17384 |

Table J.1: Query Set 1: Class and Instance Specified (Part 1)

| Instance | Class | Duration(ms) |
|---|---|---|
| copyright | proprietary notice | 11442 |
| trademark | law | 17164 |
| france | country | 3028 |
| com | site | 14805 |
| software | material | 1154 |
| email address | contact information | 5532 |
| diabetes | chronic disease | 26540 |
| child | family | 16687 |
| cat | animal | 17214 |
| facebook | social networking site | 673 |
| age | factor | 901 |
| diabetes | disease | 865 |
| computer | device | 39255 |
| facebook | site | 545 |
| name | identifiable information | 1550 |
| content | material | 1246 |
| email address | information | 1171 |
| copyright | notice | 10937 |
| new york | city | 7372 |
| facebook | social media site | 489 |
| book | material | 1089 |
| brazil | country | 13365 |
| loss of principal invested | investment risk | 8980 |
| human | animal | 8082 |
| tablet | mobile device | 1157 |
| fact | amusement account | 11790 |
| world | better place | 18733 |
| video | material | 1251 |
| potential for loss | risk | 255 |
| illinois | member fdic | 110 |
| state farm bank | member fdic | 7 |
| illinois | equal housing lender | 8382 |
| state farm bank | equal housing lender | 410 |
| cookie | text file | 12976 |
| uk | country | 2831 |
| cookie | technology | 19735 |
| friend | people | 23362 |
| pain | symptom | 8564 |
| usa | country | 2602 |
| breakfast | meal of the day | 27022 |
| advertising | use | 5937 |
| twitter | social network | 9589 |
| bird | animal | 5914 |
| saturday | day | 8323 |
| football | sport | 12090 |
| mexico | country | 2922 |
| there | of people | 648 |
| tablet | device | 17927 |
| website | resource | 55047 |
| time | thing | 26568 |
| child | family member | 13914 |

Table J.2: Query Set 1: Class and Instance Specified (Part 2)

| Class | Results | Duration(ms) |
|---|---|---|
| list | 95130 | 21903 |
| information | 248593 | 42434 |
| worldwide leader in virtualization | 17 | 16200 |
| automation software | 119 | 20959 |
| delivery | 8119 | 29924 |
| language | 57107 | 15968 |
| country | 192758 | 92091 |
| personal information | 10327 | 6297 |
| whous privacy protection service | 2 | 31016 |
| people | 364940 | 22513 |
| long time | 38906 | 18399 |
| search engine | 15920 | 13255 |
| virtue | 15344 | 8537 |
| day | 153446 | 14192 |
| thing | 901381 | 35671 |
| disease | 75205 | 23689 |
| social media | 17470 | 34606 |
| link | 133828 | 16068 |
| form of flattery | 3483 | 19834 |
| good day | 12980 | 3169 |
| resounding yes | 3008 | 4413 |
| drug | 55696 | 9574 |
| example | 337762 | 29552 |
| social network | 10703 | 12172 |
| proprietary rights | 1481 | 10076 |
| family member | 27003 | 18246 |
| medicine | 22183 | 2106 |
| animal | 78780 | 12985 |
| material | 182169 | 35797 |
| harmful component | 315 | 65491 |
| expense | 37179 | 9016 |
| website | 80727 | 8213 |
| financial institution | 13206 | 41463 |
| time | 367420 | 12865 |
| journey | 23112 | 5611 |
| policy | 47050 | 19831 |
| proprietary notice | 284 | 12522 |
| law | 57998 | 16433 |
| site | 151157 | 20716 |
| contact information | 4744 | 12820 |
| chronic disease | 5919 | 28387 |
| family | 156067 | 24870 |
| social networking site | 4286 | 3433 |
| factor | 270784 | 11952 |
| device | 173534 | 35371 |
| identifiable information | 1940 | 6368 |
| notice | 11168 | 537 |
| city | 109967 | 9117 |
| social media site | 3576 | 3591 |
| investment risk | 292 | 9658 |
| mobile device | 14232 | 4983 |

Table J.3: Query Set 2: Queries with Classes Specified (Part 1)

| Class | Results | Duration(ms) |
|---|---|---|
| amusement account | 81 | 17422 |
| better place | 8298 | 28306 |
| risk | 68108 | 1654 |
| member fdic | 148 | 136 |
| equal housing lender | 438 | 12535 |
| text file | 3267 | 19151 |
| technology | 105429 | 20088 |
| symptom | 44743 | 12212 |
| meal of the day | 1699 | 33697 |
| use | 77464 | 14231 |
| sport | 45476 | 15887 |
| of people | 59398 | 23324 |
| resource | 126238 | 52093 |
| state | 125429 | 49497 |
| everyone | 114830 | 9838 |
| success | 88684 | 20466 |
| summary | 17761 | 4046 |
| island | 30455 | 7844 |
| condition | 118453 | 89673 |
| item | 189407 | 6512 |
| resounding no | 2161 | 10775 |
| adult | 23871 | 13876 |
| content | 46389 | 12212 |
| company | 257685 | 21066 |
| different story | 42138 | 13276 |
| cost | 63056 | 6886 |
| continent | 8547 | 3286 |
| place | 394954 | 23434 |
| issue | 308618 | 9371 |
| best defense | 6327 | 33502 |
| institution | 70148 | 50975 |
| business | 147709 | 15265 |
| gift | 80810 | 5091 |
| site feature | 416 | 12455 |
| event | 233663 | 13481 |
| necessity | 54308 | 6874 |
| tablet | 15183 | 7354 |
| listing | 16084 | 15036 |
| sin | 24708 | 10680 |
| pet | 18886 | 14737 |
| mineral | 18446 | 10821 |
| organization | 180798 | 16727 |
| wildlife | 19155 | 8283 |
| picture | 67778 | 12322 |
| problem | 352855 | 68306 |
| great day | 7676 | 16058 |
| video | 66955 | 12935 |
| religion | 35908 | 61732 |
| purpose | 73070 | 5219 |

Table J.4: Query Set 2: Queries with Classes Specified (Part 2)

| Instance | Results | Duration(ms) |
|---|---|---|
| below | 204768 | 18000 |
| name | 119504 | 9623 |
| parallel | 5674 | 18826 |
| service | 108763 | 34343 |
| english | 52519 | 8907 |
| united states | 91964 | 33223 |
| following | 163662 | 13549 |
| china | 104798 | 21417 |
| whoisguard privacy protection service | 1 | 3754 |
| child | 231710 | 5993 |
| year | 302503 | 8725 |
| google | 60656 | 5764 |
| patience | 11851 | 19099 |
| canada | 55846 | 27175 |
| india | 63972 | 28880 |
| today | 175462 | 12407 |
| free lunch | 762 | 2074 |
| cancer | 71717 | 4876 |
| facebook | 75198 | 11540 |
| imitation | 3984 | 5472 |
| twitter | 56617 | 2233 |
| japan | 55946 | 4065 |
| us | 119242 | 4889 |
| answer | 53785 | 13318 |
| germany | 50467 | 8042 |
| alcohol | 43177 | 8628 |
| address | 28743 | 6851 |
| australia | 41257 | 6970 |
| trademark | 11231 | 21269 |
| parent | 78279 | 33389 |
| laughter | 8307 | 17806 |
| dog | 65839 | 13146 |
| information | 121325 | 34903 |
| virus | 21441 | 11938 |
| reasonable attorney fee | 1169 | 8507 |
| com | 203624 | 63273 |
| bank | 51109 | 19144 |
| people | 310950 | 17940 |
| there | 364838 | 22788 |
| life | 166566 | 20251 |
| product | 110939 | 42135 |
| honesty | 9487 | 20217 |
| woman | 183856 | 12789 |
| copyright | 15212 | 3824 |
| france | 41136 | 10017 |
| software | 43094 | 17325 |
| email address | 8950 | 9912 |
| diabetes | 40084 | 23840 |
| cat | 43241 | 28185 |
| age | 53572 | 4911 |
| computer | 78688 | 5846 |

Table J.5: Query Set 2: Queries with Instances Specified (Part 1)

| Instance | Results | Duration(ms) |
|---|---|---|
| content | 43175 | 7262 |
| new york | 63468 | 2463 |
| book | 280512 | 14495 |
| brazil | 29497 | 10585 |
| loss of principal invested | 13 | 14919 |
| human | 36649 | 4994 |
| tablet | 32280 | 10455 |
| fact | 260976 | 15117 |
| world | 116841 | 5892 |
| video | 159122 | 12581 |
| potential for loss | 40 | 21210 |
| illinois | 10731 | 1834 |
| state farm bank | 13 | 16447 |
| cookie | 22865 | 21265 |
| uk | 52707 | 1420 |
| friend | 118090 | 8798 |
| pain | 43436 | 31729 |
| usa | 36601 | 7812 |
| breakfast | 32295 | 1975 |
| advertising | 23179 | 10380 |
| bird | 38763 | 10857 |
| saturday | 32816 | 18004 |
| football | 30845 | 15751 |
| mexico | 28515 | 20389 |
| website | 72562 | 21903 |
| time | 358950 | 23381 |
| russia | 29578 | 5119 |
| california | 34005 | 22801 |
| me | 128249 | 1931 |
| event | 130307 | 7183 |
| sunday | 39310 | 13342 |
| man | 180514 | 33510 |
| other party for marketing | 39 | 7820 |
| smartphone | 23269 | 2753 |
| owner of website | 107 | 6081 |
| photograph | 33297 | 7815 |
| image | 93232 | 9143 |
| however | 368911 | 18831 |
| italy | 30027 | 2572 |
| family | 131283 | 19317 |
| summer | 51577 | 2426 |
| america | 57483 | 8372 |
| antarctica | 4977 | 13296 |
| home | 151067 | 6263 |
| money | 91467 | 23825 |
| spain | 25862 | 18411 |
| here | 109876 | 16537 |
| good offense | 323 | 2793 |
| school | 120422 | 11394 |

Table J.6: Query Set 2: Queries with Instances Specified (Part 2)

## Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Masterarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 30.10.2014 Unterschrift