Charles University in Prague

Faculty of Mathematics and Physics

**MASTER'S THESIS**

Bc. Jakub Kúdela

# Mining Parallel Corpora from the Web

Department of Software Engineering

| | |
|---|---|
| Thesis supervisor: | Doc. RNDr. Irena Holubová, Ph.D. |
| Thesis consultant: | RNDr. Ondřej Bojar, Ph.D. |
| Study programme: | Computer Science (N1801) |
| Specialization: | Software Systems (2612T043) |

Prague 2016

First, I would like to thank my supervisor Doc. RNDr. Irena Holubová, Ph.D. for inspiring me when selecting the topic related to my interests, for her valuable and professional advice, and for the time she has spent helping me with the thesis. I would also like to express my gratitude to my consultant RNDr. Ondřej Bojar, Ph.D. for introducing me to the domain of natural language processing and for his time and ideas, which helped me a lot. I am very greatful to my parents Libuša and Ivan for their determination to support me as much as possible. Next, I would like to thank my brother Lukáš, who has always motivated me as a great older sibling. Last but not least, I am thankful to my sweetheart Zuzana for her patience and everything she does for me.

Once again, thank you!

V prvom rade chcem pod'akovat' mojej vedúcej Doc. RNDr. Irene Holubovej, Ph.D. za inšpiráciu pri výbere témy blízkej mojím záujmom, za jej cenné a odborné rady, ochotu a čas ktorý mi v priebehu písania práce venovala. Rovnako by som rád pod'akoval môjmu konzultantovi RNDr. Ondřejovi Bojarovi, Ph.D. za uvedenie do problémov domény spracovania prirodzeného jazyka, za jeho čas a nanápady, ktoré mi v práci nesmierne pomohli. Velká vd'aka patrí mojim rodičom Libuši a Ivanovi za ich velké odhodlanie podporit' ma v živote najviac ako sa len dá. Taktiež chcem pod'akovat' svojmu bratovi Lukášovi, ktorý ma vždy v mnohom motivoval ako skvelý starší súrodenec. V neposlednom rade d'akujem mojej priatel'ke Zuzane, za jej trpezlivost' a za všetko čo pre mňa robí.

Ešte raz Vám všetkým d'akujem!

Title: Mining Parallel Corpora from the Web
Author: Bc. Jakub Kúdela
Author's e-mail address: jakub.kudela@gmail.com
Department: Department of Software Engineering
Thesis supervisor: Doc. RNDr. Irena Holubová, Ph.D.
Supervisor's e-mail address: holubova@ksi.mff.cuni.cz
Thesis consultant: RNDr. Ondřej Bojar, Ph.D.
Consultant's e-mail adress: bojar@ufal.mff.cuni.cz

Abstract: Statistical machine translation (SMT) is one of the most popular approaches to machine translation today. It uses statistical models whose parameters are derived from the analysis of a parallel corpus required for the training. The existence of a parallel corpus is the most important prerequisite for building an effective SMT system. Various properties of the corpus, such as its volume and quality, highly affect the results of the translation. The web can be considered as an ever-growing source of considerable amounts of parallel data to be mined and included in the training process, thus increasing the effectiveness of SMT systems. The first part of this thesis summarizes some of the popular methods for acquiring parallel corpora from the web. Most of these methods search for pairs of parallel web pages by looking for the similarity of their structures. However, we believe there still exists a non-negligible amount of parallel data spread across the web pages not sharing similar structure. In the next part, we propose a different approach to identifying parallel content on the web, not dependent on the page structure comparison at all. We begin by introducing a generic method for bilingual document alignment. The key step of our method is based on the combination of two recently popular ideas, namely the bilingual extension of the word2vec model and the locality-sensitive hashing. With the method applied to the task of mining parallel corpora from the web, we are able to effectively identify pairs of parallel segments (i.e. paragraphs) located anywhere on the pages of a web domain, regardless of their structure. The final part of our work describes the experiments conducted with our method. One experiment uses pre-aligned data, and its results are evaluated automatically. The other experiment involves real-world data provided by the Common Crawl Foundation and presents a solution to the task of mining parallel corpora from a hundreds of terabytes large set of web-crawled data. Both experiments show satisfactory results, implying that our proposed method is a promising baseline for acquiring parallel corpora from the web. We believe the amount of parallel data obtainable with our method might enable SMT systems to get trained better and eventually achieve superior translation results.

Keywords: mining parallel corpora, bilingual document alignment, word2vec, locality-sensitive hashing

Názov: Rafinácia paralelných korpusov z webu
Autor: Bc. Jakub Kúdela
E-mailová adresa autora: `jakub.kudela@gmail.com`
Katedra: Katedra Softwarového Inženýrství
Vedúci práce: Doc. RNDr. Irena Holubová, Ph.D.
E-mailová adresa vedúceho: `holubova@ksi.mff.cuni.cz`
Konzultant práce: RNDr. Ondřej Bojar, Ph.D.
E-mailová adresa konzultanta: `bojar@ufal.mff.cuni.cz`

Abstrakt: Štatistický strojový preklad (SMT, statistical machine translation) je v súčasnosti jeden z najpopulárnejších prístupov ku strojovému prekladu. Tento prístup využíva štatistické modely, ktorých parametre sú získané z analýzy paralelných korpusov potrebných pre tréning. Existencia paralelného korpusu je najdôležitejšou prerekvizitou pre vytvorenie účinného SMT prekladača. Viaceré vlastnosti tohto korpusu, ako napríklad objem a kvalita, ovplyvňujú výsledky prekladu do značnej miery. Web môžeme považovať za neustále rastúci zdroj značného množstva paralelných dát, ktoré môžu byť rafinované a zahrnuté do trénovacieho procesu, čím môžu zdokonaliť výsledky SMT prekladača. Prvá časť práce sumarizuje niektoré z rozšírených metód pre získavanie paralelného korpusu z webu. Väčšina z metód hľadá páry paralelných webových stránok podľa podobnosti ich štruktúr. Veríme však, že existuje nezanedbateľné množstvo paralelných dát rozložených na webových stránkach, ktoré nezdieľajú podobnú štruktúru. V ďalšej časti predstavíme iný prístup ku identifikácii paralelného obsahu na webe. Tento prístup vôbec nezávisí na porovnávaní štruktúr webových stránok. Najskôr si predstavíme generickú metódu pre bilingválne zarovnanie dokumentov. Kľúčová časť našej metódy je postavená na kombinácii dvoch súčasne populárnych myšlienok, menovite bilingválneho rozšírenia modelu word2vec a lokálne-senzitívneho hašovania (locality-sensitive hashing). S metódou aplikovanou na úlohu získavania paralelných korpusov z webu, sme schopní efektívne identifikovať páry paralelných častí (paragrafov) nachádzajúcich sa na ľubovolných stránkach webovej domény bez ohľadu na štruktúru stránok. V poslednej časti naša práca opisuje experimenty vykonané s našou metódou. Prvý experiment využíva vopred zarovnané dáta a jeho výsledky sú vyhodnotené automaticky. Druhý experiment zahŕňa reálne dáta poskytované organizáciou Common Crawl Foundation a predstavuje riešenie pre úlohu rafinácie paralelných korpusov zo stoviek terabajtov velkého množstva dát získaných z webu. Obidva experimenty ukazujú priaznivé výsledky, čo naznačuje, že navrhovaná metóda môže tvoriť nádejný základ pre nový spôsob získavania paralelných korpusov z webu. Veríme, že množstvo paralelných dát, ktoré je naša metóda schopná získať by mohlo zabezbečiť SMT prekladačom objemnejší tréning a tým pádom aj lepšie výsledky.

Kľúčové slová: rafinácia paralelných korpusov, bilingválne dokumentové zarovnanie, word2vec, lokálne-senzitívne hašovanie

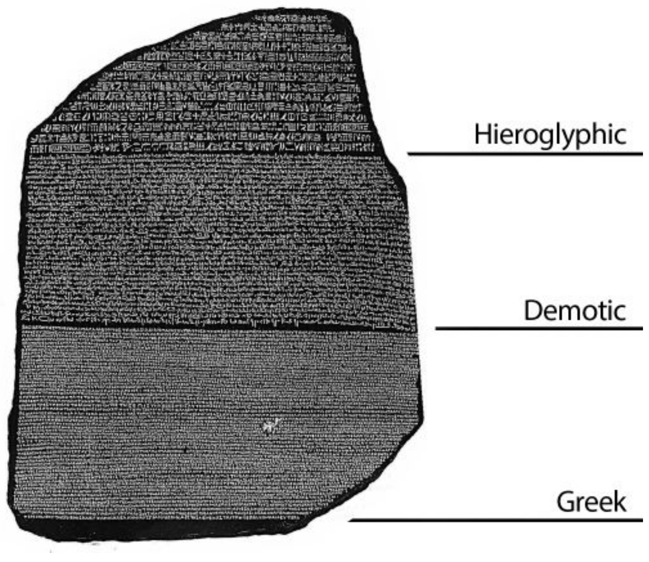
# Contents

# Introduction

Statistical machine translation (SMT) [1] is the most popular machine translation (MT) paradigm today. This approach to MT is preferred in organizations like Google or Microsoft, which play significant roles in an applied field where heavily used translation systems are deployed for the whole world to use. SMT utilizes statistical models for the translation. These models are trained on vast amounts of both *monolingual data* and *parallel data*, i.e. texts translated by humans.

The monolingual data help the system to understand what the target language should look like while the parallel data help the system to learn to translate smaller segments of one language into another. Monolingual data, also known as *text corpora*, can be described as a simple text in a single language. Parallel data, also known as *parallel text corpora*, *bilingual text corpora* (in the bilingual case) or simply *bitext*, are collections of sentence pairs, the sentences being of different languages, where one sentence is a translation of the other. One example of parallel texts in the human history is the famous Rosetta Stone (see Figure 1). The discovery of this artifact, which contained the same decree inscribed in Hieroglyphic, Demotic and Greek, led to deciphering and understanding of the extinct Hieroglyphic writing.

Figure 1: Rosetta Stone



The existence of parallel data is the most important prerequisite for building an effective SMT system. Today there exist a plenty of both monolingual and sentence-aligned parallel data. Such a collection of corpora is for example OPUS [2]. One of the richest sources of the parallel data included in OPUS are texts produced by government organizations, such as the European Parliament Proceedings, which are being translated into each of the 23 official languages of the European Union. Other important sources we could mention are movie subtitles, books, news websites, or localization files of computer software.

We believe that the web is a potential source of considerable amounts of parallel data generated not only by well-known organizations like the European Parliament but also by organizations of smaller sizes and by individuals. In the field of SMT there is also motivation and need for acquisition of parallel data from various domains. To illustrate this motivation, an SMT system trained on just the parallel data from the European Parliament Proceedings may seem cumbersome in translation of standard speech language because of the formal nature of the training data. We assume that by mining parallel data from the web, we can obtain useful parallel corpora which would also reflect the natural distribution of domains.

## Goals

The main goal of our thesis is to propose a method for solving the task called *bilingual document alignment* applicable in the field of mining parallel data from the web. This problem can be generally stated as follows: assume we have a set of documents written in two different languages, where a document is a plain text not limited by length; it can be a sentence, a sequence of multiple sentences, or even a single word. The solution of this task is a collection of all such pairs of documents in different languages that are mutual translations of each other.

We would like to distinguish our method by taking a different approach than the one shared by many others. The majority of methods developed in the field primarily use the structure of the web pages in the alignment process. We believe that by performing structure similarity filtering, we can lose a considerable part of parallel data.

The aim is to propose a more generic method, not based on page structure comparison. The method is expected to find bitexts located on pages not at all similar in structure. In other words, our motivation is to find the parallel segments, i.e. paragraphs we consider as input documents to our method, contained on a given bilingual web domain, regardless of the structure of its web pages. By moving to these finer units, we also have to rely more on the actual content of the paragraphs. To overcome the well-known problems of data sparseness, we plan to use currently popular models like *word2vec* [3][4][5], and to deal with the possibly large amount of input documents, we intent to make use of recently studied strategies for *locality-sensitive hashing* [6][7].

Any finer alignment of the documents, such as sentence alignment, is beyond the scope of our work. The methods for achieving sentence alignment for a document-aligned parallel corpus are well examined [8] and can be easily applied to the output of our method.

Moreover, our particular requirements for the target method are scalability and reasonable behavior on noisy and sparse data, where only some of the input documents will have a pair. Both these requirements are needed when processing data from the web. It is also important to mention that the method is expected to be supervised, which means it depends on the provided a priori knowledge in the form of an already existing sentence-aligned bilingual text corpus that is used in the training process. However, this does not mean that the training corpus needs to cover every single word contained in the actual input documents.

Another goal of this work is to perform experiments with the proposed method and present their results. Our experiments are solely focused on the Czech–English [9] language pair. The first experiment is carried out using CzEng 1.0 [10]—a Czech–English sentence-aligned parallel corpus. This way we can measure the quality of the results automatically. The second experiment uses more realistic, noisy data provided by the Common Crawl Foundation [11]—a non-profit organization with the goal of democratizing access to web information. It produces and maintains an open repository of web-crawled data that is universally accessible and analyzable.

## Outline

The rest of this thesis is organized as follows. Chapter 1 introduces the related work in this field, covering similar approaches that have been already explored. They can still serve as an inspiration for the potential extension of ideas covered in the thesis. In Chapter 2 we discuss technologies used in our work, together with reasoning and motivation for their selection. Chapter 3 contains a detailed description of the proposed method with all implementation details. Chapter 4 describes the experiment done on the CzEng 1.0 corpus. In Chapter 5 the experiment done on the Common Crawl dataset is explained, along with a way how to mine parallel corpora from the vast amounts of real-world web data. The final chapter concludes the discussion and suggests potential directions of future work.

# Chapter 1

# Related Work

This chapter describes methods that are the closest to ours. All approaches mentioned in the following text have one thing in common—their goal is to harvest parallel corpora from the web.

## 1.1 Bitextor

Bitextor[1] [12][13] is a well maintained free and open-source bitext generator which obtains its base corpora from the Internet. First, it downloads an entire website, keeping only the files in the plain text, HyperText Markup Language (HTML) [14] or Extensible Markup Language (XML) [15] format. Then the system detects the language of each file and applies a group of heuristics to find possible candidate pairs of files which can have the same content. These heuristics include for example file size comparison, HTML tag structure edit distance, and text blocks length difference. Once it has identified the candidate pairs of files, it generates a bitext output in the Translation Memory eXchange (TMX) [16] format.

Since the first release of Bitextor, the system has evolved and has been extended with features like heuristics based on Uniform Resource Locator (URL) and utilization of dictionaries with the bag-of-words model to help the candidate filtering process. The following text describes the current state of the core pipeline of Bitextor according to the standard workflow.

[1]`https://sourceforge.net/projects/bitextor/` (accessed March 20, 2016)

### 1.1.1 Bitextor: Procedure

1. Bitextor downloads all text/HTML/XML files from a website provided by seed URLs, while keeping their folder structure. For this purpose it internally uses a crawler called HTTrack[2].

2. It extracts encoding, MIME type and Base64-encoded content of each downloaded file. The library called boilerpipe[3] helps to remove menus and other unwanted parts from the HTML files. Then the tool Apache-Tika[4] is used to repair the structure of the HTML files, obtaining XHTML [17] files. This step also normalizes encoding of the files into uniform UTF-8.

3. The system identifies the language of the files. This step is done using langid.py[5] [18], which is a standalone language identification tool pretrained for a large number of languages.

4. A so-called *raspa* representation is created for every HTML file. In order to create such a representation every text block is replaced by the sequence of apostrophes, of length $\log_2(\text{len}(\text{text}))$, where len(text) is the length (in characters) of the given text block. This means that the representation preserves the HTML tags in-place, only the text blocks are changed. These raspa representations will later help the system to calculate the *fuzzy-matching score* used to rank the parallel file candidates.

5. The next step uses the bag-of-words from each file and a provided dictionary to compute an overlapping score between the files in both languages. For each file in one language, it creates a list of ten highest scoring candidates in the other language. The default dictionary contains English, Spanish, Catalan, Croatian, and Basque languages.

   However, the system provides a script for creating a custom dictionary. This script gets a sentence-aligned bilingual corpus in the languages we want to consider, lowercases it, and filters out overly long sentences. Then it runs GIZA++[6], a word-alignment tool that implements the series of IBM Models introduced by Brown et al. [19]. Bitextor lets GIZA++ dump parameters of the trained statistical models for both directions (one language as the source and the other as the target, and vice versa). Finally, each word pair appearing in both directions and having the value of harmonic mean of parameters great enough is included into the dictionary.

6. With the candidate lists created, the system recalculates the scores provided by the previous step. It multiplies the score of each candidate by the fuzzy-matching score between the raspa representations of the file and the candidate. The fuzzy-matching score compares two raspa sequences on

[2]`http://www.httrack.com/` (accessed March 20, 2016)
[3]`https://github.com/misja/python-boilerpipe` (accessed March 20, 2016)
[4]`http://tika.apache.org/` (accessed March 20, 2016)
[5]`https://pypi.python.org/pypi/langid` (accessed March 20, 2016)
[6]`http://www.statmt.org/moses/giza/GIZA++.html` (accessed March 20, 2016)

the basis of Levenshtein's edit distance. To calculate the edit distance the system internally uses the python-Levenshtein[7] library.

7. Bitextor allows to create the candidate lists in both directions (one language as the source and the other as the target, and vice versa). In this situation, the score for a file pair is calculated as an average of scores for both directions. If the file pair scores below the threshold in either of the directions it is discarded altogether.

8. All aligned files are first concatenated and stored in two new files, one for each language. Concatenation is done in the same order, while using a system-specific separator. The presence of separators ensures that segment alignment will happen only between the aligned files. Then the system runs hunalign[8] [20] to get the segment alignment. In the first pass, it uses Gale-Church [21] sentence-length algorithm for the alignment. Next, it creates an automatic dictionary from the first alignment. Then it realigns the text in the second pass, using the automatic dictionary.

9. After running hunalign, the system removes all unaligned segments and those with a confidence score below a specified threshold. Moreover, the system allows a configuration in which the aligned pairs of segments coming from a pair of files with too many problems are discarded. Finally, with the segments aligned and filtered, Bitextor can, optionally, output them in the TMX format.

### 1.1.2 Bitextor: Results

The system's authors suggest favoring precision over recall. They performed an experiment covering two websites. The system's output was compared against human evaluations. The preliminary results show a great precision of approximately 100% at a reasonable recall of more than 63%.

### 1.1.3 Bitextor: Summary

We consider Bitextor as a mature, language independent tool which can handle many typical scenarios when facing the problem of harvesting parallel corpora from multilingual websites. Yet its functionality is based on a premise that parallel texts are stored in files with very similar HTML structure. However, there are many websites which are designed in such way that all the information is available in the primary language, with only a subset translated into secondary languages. As an example, consider a news website where each article is written in the primary language, starting with a summary followed with a body. But each of these articles has a translation available only for the summary part. This

[7]`https://pypi.python.org/pypi/python-Levenshtein/` (accessed March 20, 2016)
[8]`http://mokk.bme.hu/resources/hunalign/` (accessed March 20, 2016)

means that the page with the article written in the primary language has a different HTML structure than the page containing just the translated summary. They probably differ in the number of paragraph tags, and thus Bitextor would discard such pair due to the difference in both HTML structure and text length. Also, in the case where a single page contains a same piece of text available in both the languages, Bitextor would not find such bitext.

## 1.2 PaCo$^2$

PaCo$^2$ [22] is a fully automated tool for gathering parallel corpora from the web. Unlike others, it is also focused on identification of potential websites containing bilingual content for chosen languages. Being language independent, it is adaptable to any set of languages. The architecture of PaCo$^2$ is designed as a pipeline consisting of three main phases. In the first, phase the web is searched for websites which might contain bilingual textual content. In the second phase, parallel web pages are detected on the sites identified in the previous phase. In the third and final phase, parallel web pages are aligned at the sentence level and a parallel corpus is created. The details of each phase are described in the list below.

### 1.2.1 PaCo$^2$: Procedure

1. PaCo$^2$ handles bilingual website detection. This custom capability is a relatively unique feature in the field. It has also been implemented in the previous approaches (e.g. STRAND); however, they were entirely dependent on the AltaVista search engine, which had an application programming interface (API) that allowed to search for multilingual websites. Unfortunately, it is no longer available.

   The system uses search engines for identification of potential bilingual websites. As an input, it requires a language pair and a list of fifty or more lemmatized words from the preferably less frequent of the two languages. It then generates queries for the search engine. The queries are triplets from the list of provided words. When querying the search engine, the system uses its API to request the results for the other language. This way, a list containing URLs of candidate websites is created.

2. The preliminary filter is applied to discard a priori unwanted candidates, such as gigantic websites, blog platforms, and social media. Authors have found out that these would bring a very negligible benefit. For each of the URLs left in the filtered list, the system downloads a very small part of the website and searches for elements of parallelism, e.g. hyperlinks with anchor texts or `alt` attribute texts indicating presence of languages. If the website passes the set of requirements on this smaller scale it is considered as a website with parallel content.

3. For each website found in the previous step, the system harvests all of its HTML files using Wget[9]. In order to compare each pair of the $N$ acquired files, the system would need to perform $N^2$ comparisons in the worst case. However, various heuristics and statistical filters are applied to reduce this number. These include language, file size, and length comparison. The system compares only HTML files written in the exact two languages requested by the user. It also excludes all file pairs outside a certain similarity threshold with regard to the file size and length ratio. Language identification is performed using TextCat[10], and character encoding is standardized to UTF-8 by means of BeautyfulSoup[11].

   Authors of PaCo$^2$ decided to keep boilerplate[12] parts because they consider the menus and other elements such as breadcrumbs to contain useful parallel information. In addition, each content having the same value regardless the language of the text (e.g. copyright notes or contact information) is removed from the results. The bitext detection module runs the following three major filters when searching for parallel files:

   (a) *Link Follower Filter* focuses on the hyperlinks located in HTML structures of the files. For example, let us assume we want to find parallel HTML files for the language pair consisting of Czech and English. If there exists an HTML file pair where the structure of the first file contains a hyperlink to the second file and vice versa, and additionally, if the links contain reasonable anchor texts like “English”, “anglicky” and “Czech”, “česky”, the filter considers the pair of files as parallel.

   (b) *URL Pattern Search* draws from the assumption that parallel web pages have usually similar URLs. They often differ only in parts representing language codes. For instance, the following two URLs referencing parallel files differ only in this manner. Czech article contains “cs” in its URL, while the English one contains “en” instead.

   ```
   http://www.mff.cuni.cz/to.cs/verejnost/konalo-se/2016-03-cenaprop/
   http://www.mff.cuni.cz/to.en/verejnost/konalo-se/2016-03-cenaprop/
   ```

   To detect such scenarios, for each pair of file, PaCo$^2$ first finds and removes the language codes from both URLs. Then it calculates the Longest Common Subsequence Ratio (LCSR). If the ratio reaches a required threshold the system considers the candidates as parallel files.

   (c) *HTML Structure and Content Filter* is useful in all other scenarios not covered by the previous two filters. The system compares HTML tag structure information [23] combined with content similarity. PaCo$^2$ authors observed that by comparing only the HTML structures of the files the results are poor due to the existence of numerous files with similar HTML structures on an ordinary website.

[9] `https://www.gnu.org/software/wget/` (accessed March 27, 2016)
[10] `http://www.let.rug.nl/vannoord/TextCat/` (accessed March 27, 2016)
[11] `http://www.crummy.com/software/BeautifulSoup/` (accessed March 27, 2016)
[12] surplus “clutter” (templates) around the main textual content of a web page

In order to compare content similarity, the system works with extracted universal entities, e.g. numbers, email addresses and dates. The algorithm for extraction of these language independent entities has been adapted from the previous work in the field [24]. To compute the content similarity of candidate parallel files, vector representation consisting of extracted entities is prepared for each file. The order of entities corresponds to the order of their appearance in the content. The Ratcliff/Obershelp [25] pattern recognition algorithm is then used to compare these vectors.

To calculate the total score for a given pair of candidate parallel files, their HTML structure similarity and content similarity is calculated. Both results are weighed, and those candidates that reach a certain threshold are regarded as parallel files. If there are multiple candidates that have reached the threshold to be a parallel file with one file, all of them are ruled out.

4. Sentence alignment is done with the help of the hunalign tool. Then, a series of post processing activities is applied. Translation units (i.e. pairs of sentences) are sorted and deduplicated. Units containing solely elements like numbers, email addresses, or URLs are excluded. Language identification is done once again, but this time at the sentence level. Units that do not fit the requested languages are discarded. At the end of the process, if there are more than two unique target texts for the same source text, all units containing this source text are omitted in favor of precision. The output can be either in raw text or in the TMX format.

### 1.2.2 PaCo$^2$: Results

PaCo$^2$ authors report experiments on these language pairs: Basque–Spanish, Spanish–English and Portuguese–English. In order to identify the bilingual websites, PaCo$^2$ utilized the Bing[13] search engine. Results show acceptable precision of 97% for the Basque–Spanish language pair. However, the precision levels for the Spanish–English and Portuguese–English pairs were relatively poor, 67% and 78% respectively. Error analysis over the wrong candidates showed the accuracy of the system to be low due to the fact that many of the web pages covered in the experiment contained dynamically variable elements (e.g. tag-clouds).

### 1.2.3 PaCo$^2$: Summary

PaCo$^2$ is a language independent system. Its unique feature of identification of websites containing parallel data is beneficial. The aligning process is entirely unsupervised, which we also consider a valuable feature. The only a priori knowledge required to run the whole process of parallel data acquisition is a list of fifty

[13] `http://www.let.rug.nl/vannoord/TextCat/` (accessed March 27, 2016)

or more lemmatized words from one of the two chosen languages. We appreciate the focus on performance, where PaCo$^2$ uses a set of heuristics. These identify and handle common scenarios, which leads to faster delivery of results. Unfortunately, like Bitextor, PaCo$^2$ cannot handle alignment of parallel data segments located in files with different HTML structure.

## 1.3 STRAND

STRAND [23][26][27] is a system performing structural translation recognition. Its main objective is to identify pairs of parallel web pages. Its design is based on an observation of how parallel content is usually distributed over an ordinary website. The authors suggest that websites have a very strong tendency to present parallel content via pages having a similar HTML structure. The text below describes how the system works.

### 1.3.1 STRAND: Procedure

1. At the beginning of the process, STRAND tries to locate websites containing parallel data. For this task, it used to utilize advanced search capabilities of the AltaVista search engine which, unfortunately, is no longer active. The engine helped with searching for two types of web pages:

    - A *parent* page contains hyperlinks to web pages containing different language versions of the same content. An example of such a page is one containing two links to the same article written in English and French. To perform search for such a page, the system queries the engine with an expression `(anchor:"english" OR anchor:"anglais") AND (anchor:"french" OR anchor:"français")`. Additionally, a distance filter is applied on the obtained results. If the positional distance between the links within the HTML structure is more than 10 lines, the page is filtered out. Authors believe that parent pages tend to have links pointing to the corresponding pages closely located within their HTML structures.
    - A *sibling* page is a type of page in one language that contains a hyperlink to a version of the same page in another language. A page written in French containing an anchor with a label ”English”, which points to another page with the same content in English, is an example of a sibling page. The system searches for these by querying the engine with an expression like `(anchor:"english" OR anchor:"anglais")`.

    After AltaVista search engine was shut down, authors added another component to the system called *spider*. It can download all the web pages from a provided initial list of websites.

2. Generating candidate pairs is simple when a search engine is used to acquire the list of parent and sibling pages. For each parent page, the system takes the two linked pages, and for each sibling page, the system takes the page together with the linked one.

   However, when all the pages from a website are under consideration, naturally, the task of generating candidate pairs is more difficult. The spider component contains a URL-matching stage which exploits the fact that parallel pages have usually similar URL structure. The system knows substitution rules which help to detect if a URL of a page in one language can be transformed into a URL of another page in a different language. If so, these two pages are considered as a candidate pair. Web pages that are mutual translations tend to have a very similar ratio of lengths; therefore, the system requires from a candidate pair to have a reasonable value of this ratio, otherwise it is discarded.

3. The key element of STRAND is a structural filter. It analyses HTML structure of both pages of a candidate pair and decides whether they are parallel or not. First, the system linearizes the HTML structure while ignoring the textual content of the page. As a result, it generates a linear sequence of tokens for each HTML structure. Such a sequence consist of three types of tokens:

   - `[START:element_label]`;
   - `[END:element_label]`;
   - `[Chunk:text_length]`.

   First two types of tokens are substitutes for HTML opening and closing elements. The last type of token is a substitute for a non-markup text block. The length is calculated as the number of non-whitespace bytes contained within the block. The attributes of the HTML elements are treated similarly as a non-markup block. For example `<font color="blue">` would generate a subsequence `[START:font][Chunk:12]`. These sequences, representing linearized structure of both web pages, are aligned by means of an adapted algorithm [28] based on dynamic programming. With alignment done, the system calculates 4 scalar values, which eventually determine whether the candidate pair is parallel or not. These values characterize the quality of the alignment, but also the probability of the candidate pair to be parallel:

   - *dp* The difference percentage of the non-shared tokens. It represents the ratio of mismatches in the alignment. Mismatches are tokens from one sequence not having a corresponding token in the other sequence and vice versa. The greater ratio indicates that the two web pages have different structure and therefore are less likely parallel. This can also happen when the one page contains the full article while the other contains only the translation of its introduction. The system requires the value to be less than 20%.
   - *n* The number of aligned non-markup text blocks of unequal length. The aligning algorithm maximizes the number of matches of the iden-

tical tokens which represent markup. As a side effect, it also pairs the tokens of the corresponding non-markup text blocks. The higher number of aligned pairs of the non-markup text blocks is found, the more likely it is the web pages are parallel.

- *r* The correlation of lengths of the aligned non-markup blocks. The authors assume that for each pair of parallel web pages, it is likely that the lengths of their corresponding text blocks are in linear relationship. This means that shorter text blocks of one page correspond to the shorter text blocks of the other page, and so it is with medium and longer blocks. The Pearson [29] correlation coefficient is approximately 1.0 in such scenarios, indicating positive correlation.
- *p* The significance level for the correlation of lengths of the aligned non-markup blocks. This measure describes the reliability of the previous one. The system requires the value to be less than 0.05, which corresponds to more than 95% confidence that the correlation value was not obtained by chance.

### 1.3.2 STRAND: Results

At first, the authors used a set of manually, empirically chosen thresholds for the discussed scalar values based on their observations from an experiment with English–Spanish development data. The thresholds seemed to perform well also when testing on English–French and English–Chinese data. Later, they investigated options of optimizing the binary decision task, whether to label the candidate pair of web pages as parallel or not, based on the mentioned scalar values. They came up with the idea of building a supervised classifier utilizing the C5.0[14] decision tree software. With the original manually chosen thresholds, the system achieved 68.6% recall at 100.0% precision, while using the learned classifier it achieved 84.1% recall on average at the cost of lower 95.8% precision.

### 1.3.3 STRAND: Summary

STRAND is one of the pioneers in the field of tools for parallel corpora acquisition from the web. Its authors have introduced some of the ideas still present in other similar tools. We consider the later added supervised binary classification as an effective approach.

[14]`http://www.rulequest.com/demoeula.html` (accessed April 3, 2016)

## 1.4 Mining Wikipedia

In previous related work, the researchers have developed a method [30] that can build subject-aligned comparable corpora, which are later refined to obtain truly parallel sentence pairs. The efficiency of the method is demonstrated on the Polish–English Wikipedia[15] content.

In order to better understand this method, the authors suggest distinguishing between a few types of corpora according to their properties:

- A *parallel corpus* is the most valued and rare type. It can be defined as a corpus that contains quality translations of documents in multiple languages. Such a corpus is already aligned or should be very easy to align at the sentence level.
- A *noisy-parallel corpus* contains bilingual sentences that are not perfectly aligned or the quality of the translations is poor. However, the majority of documents should be present in the corpus, including their translations.
- A *comparable corpus* consists of bilingual documents that are neither sentence-aligned nor mutual translations. Nevertheless, the documents should be at least topic-aligned.
- A *quasi-comparable corpus* includes very heterogeneous and very non-parallel documents that do not even have to be topic-aligned.

The authors have proposed a methodology which can extract a truly parallel corpus from a non-sentence-aligned one, such as a noisy-parallel or comparable. The implementation of the methodology has the form of a pipeline which includes specialized tools for obtaining, aligning, extracting, and filtering text data. This pipeline is explained in the following text.

### 1.4.1 Mining Wikipedia: Procedure

1. Web crawling is the first step in the process. The pipeline includes a specialized web crawler dedicated solely to processing the Wikipedia website. The crawler requires a hyperlink to a specific article, preferably written in the less frequent of the two languages. For the Polish–English language pair this would be Polish. With the link to the Polish article, the crawler downloads its page and also other pages in the topic domain (following the links on the pages), together with their corresponding English versions. With the HTML web pages obtained, the crawler extracts and cleans their textual content. This means that all links, figures, pictures, menus, references,

[15] `https://www.wikipedia.org/` (accessed April 3, 2016)

and other unwanted parts of the data are removed from further processing. The bilingual extracted texts are then tagged with unique IDs to form a topic-aligned comparable corpus.

2. The system uses a two-step sentence alignment method provided by hunalign. The pipeline does not provide any dictionary to hunalign. Without a dictionary, hunalign first aligns the sentences using just the Gale-Church [21] sentence-length algorithm. Then it builds an automatic dictionary based on the sentence alignment from the first run. This dictionary is used in the second run to realign the alignment, improving the result.

   Like the majority of sentence aligners, hunalign does not perform well when the order of corresponding segments is different for two languages. This happens when segment "A" is followed by "B" in one language, while in the other one "B" is followed by "A". The method deals with this problem by applying a posteriori filtering process on the results obtained from the hunalign. The goal of the filtering process is to find the correct alignment for each source sentence if such alignment exists, or to remove the sentence from the resulting corpus otherwise.

3. The filtering strategy of the pipeline is to find the correct translation for each Polish sentence using a translation engine. Given an MT system, the filtering process first translates all the Polish sentences into English. Then it uses a series of heuristics to compare the obtained machine-translated sentences with the original English sentences. The authors have considered many ways to measure the similarity of two sentences in the same language.

   One of the measures is defined as the number of common words divided by the total number of words in both sentences. Removing the stop words[16] (e.g. "a" or "the") prior to the measurement yields more precise results. When comparing the machine translated sentence with the original one, the translated one often contains a stem of a corresponding word (e.g. "boy" vs "boys"). This cannot be detected by the described measure. Another important aspect ignored by this measure is the order of words. To tackle this, the authors added another measure based on the concept of string similarity.

   Synonyms are another problem. In order to take them into account the method uses WordNet®[17] [31][32] together with the NLTK[18] [33] Python module. The method generates multiple derived sentences for each original sentence using synonyms. The generated sentences are then compared in a many-to-many relation.

   To obtain the best results, the script that covers the filtering process provides the ability to let the user choose multiple filtering heuristic functions with different acceptance rates. The faster functions, usually with lower quality results, are calculated first. If they find a result with a very high acceptance rate, their result is accepted as the final one. Otherwise, slower functions, with higher precision in general, are used in the filtering process.

[16]usually the most common words in the language with very little meaning

[17]`https://wordnet.princeton.edu/` (accessed April 3, 2016)

[18]`http://www.nltk.org/` (accessed April 3, 2016)

The filtering procedure of the method requires a translation engine. A custom one was built to be used. The training data included Polish–English parallel data for various domains from OPUS. To increase the system's precision, the authors adapted it to Wikipedia using the dump of all English content as a language model. The underlying MT system was based on Moses [34][35] toolkit.

### 1.4.2 Mining Wikipedia: Results

The conducted experiments show that the filtering method has quite good precision of more than 95%. The results also correlate with human evaluations. The method is language independent and supervised, as it needs a parallel corpus for the initial training of the SMT system. The amount of obtained data, in other words, the method's recall, is absolutely not satisfactory. Authors suggested that an iterative approach could increase the recall. In such a scenario, after each iteration, the newly acquired parallel corpus would help to build a domain-specific vocabulary for hunalign and to retrain the SMT system.

### 1.4.3 Mining Wikipedia: Summary

The described pipeline shares some common features with our method. In particular, both methods focus on extraction of truly parallel corpora from noisy data. Unfortunately, it is hard to tell to what extent it would be possible to optimize the set of chosen heuristic functions and their acceptance rates to gain better overall recall. This would require further experiments with the system, which would need to be obtained and deployed locally.

## 1.5 Mining Common Crawl

Another interesting project [36] closely related to ours is the one focused on mining parallel corpora on the web-scale from the Common Crawl [11] dataset. This work is heavily based on the previously described and discussed STRAND [23] algorithm. Using a set of common two- or three-letter language codes, this method achieved to mine parallel corpora for dozens of language pairs from 32 terabyte (TB) large dataset in a time span shorter than a single day. The amount of parallel data acquired in this way is large and the quality is reasonable. Moreover, the parallel corpora obtained cover data from various domains. These corpora provably increase the performance of machine translation systems if included in the training process.

The authors argue that any sophisticated method for mining parallel corpora from the web requires direct access to a larger dataset consisting of crawled web pages together with the computing power to process them. They claim that these large-scale web-crawled datasets were, until recently, available solely to large companies with the resources to crawl, store, and process the data from the entire web. Only recently, the Common Crawl non-profit organization began to provide a large-scale partial copy of the web to researches, companies, and individuals at no cost for research and analysis.

The Common Crawl corpus is stored and hosted by Amazon Web Services[19] as a part of Public Data Sets[20] in Simple Storage Service (S3)[21]. It can be either downloaded to a local cluster, accessed from Amazon Elastic Compute Cloud (EC2)[22], or processed using the Amazon Elastic MapReduce (EMR)[23] service. This section includes only a brief description of the Common Crawl dataset and the MapReduce [37] framework necessary for us to understand this project. The following text describes the flow of the method proposed by this project.

### 1.5.1 Mining Common Crawl: Procedure

1. The first step in the pipeline performs identification of potentially parallel pairs of web pages, using the Amazon EMR framework. This is the only step executed remotely on the Amazon servers. All the other steps are performed locally, using the downloaded dataset created in the first step. The Amazon EMR is chosen as the processing framework because the MapReduce paradigm suits the task well. For the sake of brevity, we can describe the Common Crawl corpus as a huge set of crawled and stored web pages in the form of HTML requests and their responses. To briefly introduce the MapReduce framework, it is a mechanism which allows us to iteratively process all the crawled HTML requests and responses in the corpus in a distributed manner.

   The method starts by remotely executing the MapReduce application, which implements two phases: *map* and *reduce*. The intention is to scale down the vast amount of crawled web pages to a selection of candidates possible to process locally.

   During the map phase, the method iterates over each web page entry in the corpus. It scans the URL of the page, searching for the occurrences of two types of substrings:

   - Language codes in ISO 639 format (two- or three-letter codes).
   - Language names in English and also in the language of their origin.

   If such a substring, surrounded by non-alphanumeric characters, is present in the URL the page is identified as potentially having parallel versions. In this case, the method outputs the URL of the page with the matching

[19]`https://aws.amazon.com/` (accessed April 4, 2016)
[20]`https://aws.amazon.com/public-data-sets/` (accessed April 4, 2016)
[21]`https://aws.amazon.com/s3/` (accessed April 4, 2016)
[22]`https://aws.amazon.com/ec2/` (accessed April 4, 2016)
[23]`https://aws.amazon.com/elasticmapreduce/` (accessed April 4, 2016)

substring replaced by a single asterix symbol. It also marks down the language associated with the replaced substring. For example, when processing the page with URL `http://www.ksi.mff.cuni.cz/en/`, the method would output the following key-value pair with the composite value:

- Key: `http://www.ksi.mff.cuni.cz/*/`.
- Value:
    - `http://www.ksi.mff.cuni.cz/en/` (original URL);
    - English (language associated with the code);
    - full HTML structure of the web page.

In the reduce phase, the method obtains all the values having the same key, i.e the language-independent URL. If there are at least two items associated with the same key having different languages marked down, the method outputs all the values for such a key.

Upon completion of the MapReduce execution, the resulting dataset is downloaded to a local cluster for further processing. This dataset is relatively small compared to the original one, and it can be processed locally.

2. The rest of the process is similar to STRAND (see Section1.3). In order to determine which web pages are parallel, the method linearizes their HTML structures in the same way as STRAND does. Additionally some HTML elements are ignored (e.g. `<font>` or `<a>`). The pairs of sequences are aligned using an algorithm based on dynamic programming which optimizes the number of matching tokens. The alignment is used to calculate the same set of measures defined by STRAND. The result of this step is a set of matching text blocks gathered from the pairs of corresponding web pages.

   Also inspired by STRAND, the project's authors tried to train the maximum entropy classifier for the Spanish–English language pair by using a set of 101 manually aligned and annotated pairs of web pages. However, it turned out that even when using the best performing subset of the features, the classifier did not outperform a naïve one, which considers every pair of web pages to be parallel. They argue that this was caused by the unbalanced nature of the annotated training data, where 80% of the pairs were parallel. In the end, they decided to exclude the classifier from the pipeline.

3. The pairs of matching text blocks then pass through the process of segmentation. The method uses the Punkt sentence splitter from the NLTK [33] Python module to perform segmentation at both the sentence and word level.

4. With segmentation done, each of the matching text blocks is sentence-aligned using the Gale and Church [21] algorithm. The output of the execution can be regarded as a parallel corpus.

5. Lastly, the process includes the final cleaning of the produced parallel corpus. The whole pipeline does not perform any boilerplate removal in the

previous steps. Since the authors decided not to do so, they at least proposed to remove the pairs of segments where either both segments are identical, or one of the segments appears multiple times in the corpus.

### 1.5.2 Mining Common Crawl: Results

To estimate the recall of the heuristic used for candidate selection, the same method was applied on a set of previously mined parallel pairs of URLs included in the French–English Gigaword [38] corpus. As a result, 45% of all pairs have been discovered. Inclusion of one-letter language codes (e.g. "f" for French, "e" for English) increased the recall of the method's recall to 74%, but the authors decided not to use such codes out of concern that the system might lose precision. In order to estimate the method's precision, a manual analysis was conducted, where 200 randomly selected sentence pairs were compared for 3 language pairs. For the German–English pair, 78% of the mined data represented perfect translations, 4% were paraphrases, and 18% represented misalignments. Additionally, 22% of true positives were probably machine translations, and in 13% of all true positives, one sentence contained some extra content not present in the other.

### 1.5.3 Mining Common Crawl: Summary

In terms of large-scale processing, the project of mining Common Crawl dataset is unique in the field. A special approach is needed when processing hundreds of terabytes large datasets. The Common Crawl dataset is a perfect example of Big Data [39], which are rising in popularity recently. The method is a baseline idea of how to mine parallel corpora from vast amount of web-crawled data.

## 1.6 Summary

Bitextor searches for the pairs of parallel web pages by comparing their HTML structures and contents. The structures are compared using a method based on dynamic programming. When comparing the contents, Bitextor uses bag-of-words model powered by a bilingual dictionary.

PaCo$^2$ reduces the number of comparisons by using a set of heuristics including length and file size comparison. To find pairs of parallel web pages, it applies a set of filters. These include inspecting the links between a pair's pages and similarities in their URLs. However, the most important filter compares the HTML structures and contents. Unlike Bitextor, PaCo$^2$ compares the contents by identifying the common universal entities (e.g. numbers or e-mail addresses). This approach is unsupervised and language independent.

To reduce the number of comparisons, STRAND uses heuristics similar to those in PaCo$^2$. It comes with an idea to train a decision tree model to classify pairs of web pages as parallel or not. The model uses a set of measures defined on the differences between the two HTML structures. This approach is supervised, as it needs to be trained using a set of, preferably, manually aligned web pages.

The method of mining parallel data from Common Crawl dataset uses the MapReduce framework to refine the candidates, which are then processed locally by a STRAND-like method. The web pages are considered to be candidates if they have the same URLs stripped from the language codes.

The method of mining Wikipedia takes a different approach. For a particular topic, it collects pairs consisting of an article and its version in the other language. The method assumes that the two articles may have different HTML structure and order of the textual context. The pair's articles are aligned at the sentence level, and the resulting alignments are refined with the help of a pre-trained SMT system.

All of the described methods, except the one for mining Wikipedia, search for parallel content only in the form of complete web pages with a similar HTML structure. Although the method for mining Wikipedia does not require the pairs of article pages to have similar structure, the quality of its results depends on the degree of correspondence between the articles' segments and their order.

# Chapter 2

# Background Work and Prerequisities

This chapter is an introduction to the resources and tools used in our work. It describes their features and properties and discusses their usefulness in our method or experiments. Before we introduce these individual resources and tools, let us first provide the context in which they are used by outlining the idea behind our method.

## 2.1 Overview of Proposed Method

All the methods described in Chapter 1 operate on common basic principles. They assume the parallel data are most often created in the form of complete web pages with a similar HTML structure or matching URL, and they work well under these circumstances. However, we believe that there still exists a non-negligible amount of parallel data spread across the web pages that have different HTML structure and URL. Also, the order of corresponding segments may vary in different language versions of the same page. Moreover, some parallel content can be a part of a single web page. We would like to address all these situations.

The majority of described methods start the process by identifying candidate pairs of parallel web pages to be inspected. If a candidate pair has a similar HTML structure then the segments of the two web pages are aligned according to the structure.

In contrast to these methods, our approach is more generic. The method considers as the input a set of plain text documents for both the languages. These can be all the paragraphs from a bilingual web domain. The intention is to identify candidate pairs of parallel documents. The problem is that for a given web domain, the number of possible pairs of paragraphs is usually significantly greater than the number of possible pairs of web pages. We have to reduce the number of candidate pairs of documents to be inspected as much as possible. Our method is supervised, and in order to be trained, it requires a sentence-aligned training parallel corpus for the given language pair. It begins by learning distributed word representations, i.e. word vectors, for both languages using the training corpus. These word vectors are similar for context-related words, even cross-lingually. They are used to calculate aggregate document vectors. Two parallel documents have a similar document vector. The document vectors for one language are subsequently hashed, forming a search index in such a way that similar vectors end up close to each other. The search index is then queried to obtain a short list of candidates for each document in the other language. At this stage, the complexity of the problem is significantly reduced. The method inspects the candidates to determine the best one using a bilingual dictionary. The bilingual dictionary is pre-calculated statistically using the training parallel corpus. This corpus is also used to train the binary classifier, that decides, at the end of the process, whether the pair consisting of a document and its top candidate should be considered parallel or not.

With the basic idea behind the method explained, let us describe the set of resources and tools we use in our work. The details of our method are elaborated in the next chapter.

## 2.2 CzEng 1.0

CzEng 1.0 [10] is the fourth release of a sentence-aligned Czech–English parallel corpus, freely available for non-commercial research purposes. It consists of approximately 15 million sentence pairs (233 million English and 206 million Czech tokens) from seven different types of sources. The proportion of these source domains is summarized in Table 2.1.

To download CzEng 1.0, one must first apply for the registration[1]. The corpus can be downloaded in multiple formats. In our experiments, we use all training sections (packs 00–97) of CzEng 1.0 in the plain text, untokenized format. A file in this format contains tab-separated values, where every line stands for a different sentence pair and individual columns represent:

1. Sentence pair ID;
2. Filter score (indicating the quality of the sentence pair);
3. Czech sentence (untokenized);
4. English sentence (untokenized).

Because our method is supervised, it needs a sentence-aligned parallel corpus in order to be trained. For this purpose, we use CzEng 1.0 in our experiments. In

[1]`http://ufal.mff.cuni.cz/czeng` (accessed April 5, 2016)

one of them, we also utilize the alignment of the corpus to perform the evaluation of the method's efficiency. It is important to note that when using CzEng 1.0 as both the training and testing dataset, we first split the data into two equally sized parts: *head* and *tail*. The head is used for training, while the tail is used for evaluation. The domains are not grouped together within the corpus, but they are shuffled. This means that both head and tail should contain data from each domain.

Table 2.1: CzEng 1.0 source domains distribution (Source: [10])

| **Source Domain** | **Sentences** | **Ratio (%)** |
|---|---:|---:|
| Fiction | 4,335,183 | 28.64 |
| EU Legislation | 3,992,551 | 26.38 |
| Movie Subtitles | 3,076,887 | 20.33 |
| Parallel Web Pages | 1,883,804 | 12.45 |
| Technical Documentation | 1,613,297 | 10.66 |
| News | 201,103 | 1.33 |
| Project Navajo | 33,301 | 0.22 |
| **Total** | 15,136,126 | 100.00 |

## 2.3 MorphoDiTa

MorphoDiTa [40][41] (Morphological Dictionary and Tagger) is an open-source tool performing morphological analysis of natural language texts. Its features include morphological analysis, morphological generation, tagging, and tokenization. It can be used either as a standalone tool or a library.

In our experiment, we want to determine whether lemmatization helps our method get better quality results. For these purposes, we use MorpohoDiTa. As our experiments cover only the texts from the Czech–English language pair, we use available MorphoDiTa models for both Czech [42] and English [43].

## 2.4 SyMGIZA++

SyMGIZA++ [44] is a tool for computing symmetric word alignment models. It is an extension of MGIZA++, which in turn is a successor of the historically original program called GIZA++.

GIZA++ is an implementation of the IBM Models 1–5 [19], the HMM model [45], and the Model 6 [46]. Comparison of all these models was already analysed and discussed [46]. In the alignment process, it uses an Expectation–Maximization [47] (EM) algorithm. This iterative algorithm consists of two steps that are performed in each iteration. In the first step, called the E-step (expectation step), the previously computed model (or the initial model) is applied to the data and expected counts for individual parameters are computed using the probabilities of this model. The second step, called the M-step (maximization step), then takes these expected counts as a fact and uses them to estimate the probabilities of the next model. Furthermore, prior to the training itself, GIZA++ uses the program called mkcls [48] to create word classes. The notion of these classes helps in the training process.

GIZA++ was designed as a single-threaded application. However, MGIZA++ [49] extends GIZA++ with capabilities to run the alignment process in multiple threads on a single computer. This helps the overall speed of the program and reduces the time needed to accomplish the task.

All the models provided by the original GIZA++ are asymmetric. For a chosen translation direction, they allow words to be mapped in a many-to-one, but not in a one-to-many relationship. It is, therefore, quite common and popular to train models for both translation directions and symmetrize the resulting word alignments to allow more natural relationships between words. In this scenario, models for both directions are trained independently and post-symmetrization is applied as the final step of the process. However, the authors of SyMGIZA++ argue that introducing continuous symmetrization, during the whole training process, results in a better quality alignment. Unlike its predecessors, SyMGIZA++ produces results of the symmetrized models. Moreover, it allows us to update these symmetrized models between each iteration of the original training algorithms. The general training scheme, depicted in Figure 2.1, illustrates all the moments when the model parameters are combined during the process.

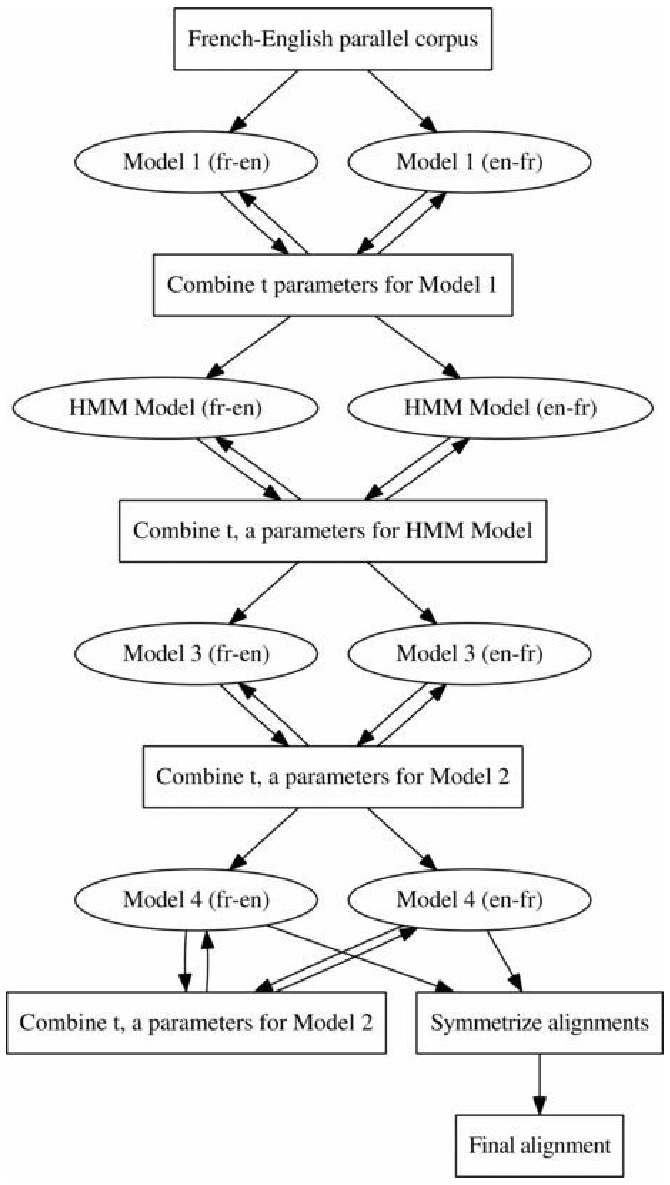
From all the available word aligning tools, such as GIZA++, MGIZA++, fast_align [50], etc., we decided to use SyMGIZA++. The results presented by its authors provide a convincing argument. They claim that the quality of the resulting alignment is increased by more than 17% when compared to MGIZA++ or GIZA++, probably the most popular word alignment programs nowadays.

To train bilingual word vectors we need a parallel corpus together with its word alignment. In order to get the word alignment for the provided training corpus, we use SyMGIZA++. Furthermore, when running SyMGIZA++, the user can request the final values of the IBM Model 1 “t” parameters for both directions. In our method, these values are used to build the bilingual dictionary.

## 2.5 bivec

*Word embedding* is a common name for a set of techniques which map words or phrases from a vocabulary to distributed word representations in the form of vectors consisting of real numbers in a high-dimensional continuous space. This section discusses bivec [51], a word embedding tool that creates bilingual word

Figure 2.1: General training scheme for SyMGIZA++ (Source: [44])

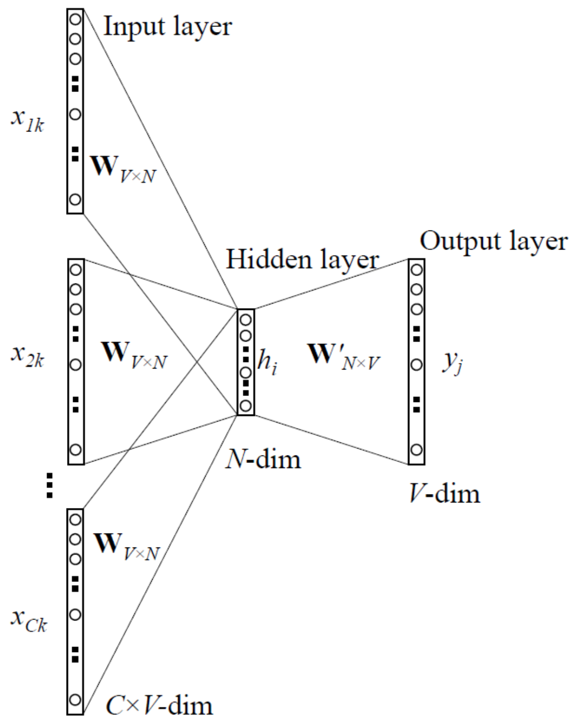

French-English parallel corpus
Model 1 (fr-en)
Model 1 (en-fr)
Combine t parameters for Model 1
HMM Model (fr-en)
HMM Model (en-fr)
Combine t, a parameters for HMM Model
Model 3 (fr-en)
Model 3 (en-fr)
Combine t, a parameters for Model 2
Model 4 (fr-en)
Model 4 (en-fr)
Combine t, a parameters for Model 2
Symmetrize alignments
Final alignment

representations when provided with a word-aligned parallel corpus. The work is an extension of previous research, which resulted in a tool called word2vec [3][4][5].

The original word2vec is a group of models producing monolingual word embeddings. These models are implemented as neural networks. They are trained to reconstruct the contexts of words. A monolingual corpus is needed for the training. During the training process, the algorithm iterates over the words in the corpus while considering the context of the current word in the form of a fixed-size window on its surrounding words. The two included models are:

- The *Continuous Bag-of-Words (CBOW)* model, trained to predict the word when given its context (without the current word).
- The *Skip-gram (SG)* model, trained to predict the context of a given word.

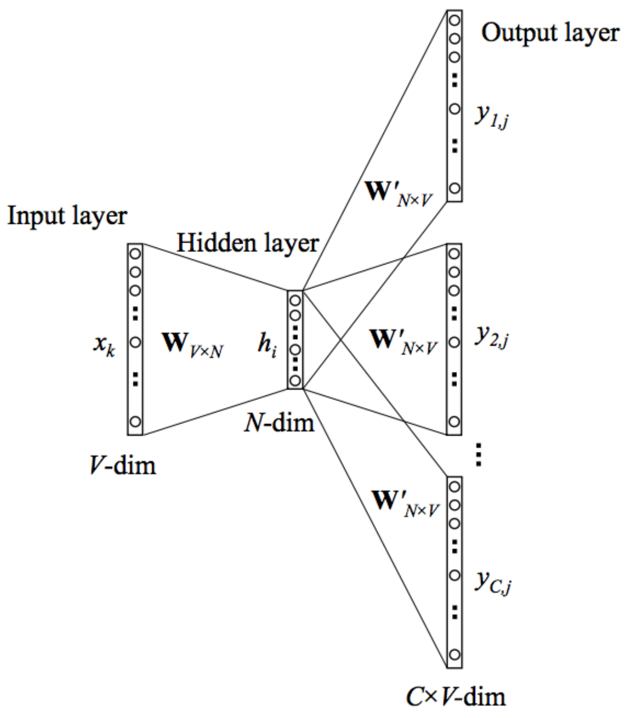
Figure 2.2: Continuous Bag-of-Words model (Source: [52])



The main difference between these two models can be observed in Figure 2.2 and Figure 2.3, which illustrate the structures of their underlying neural networks. When working with these networks, all the words are encoded to so-called one-hot vectors. This means that if $w_0, w_1, \ldots, w_V$ are all the unique words of the training corpus, word $w_i$ is encoded to a $V$-dimensional vector, where the $i$-th element is 1 and all other 0s.

In Figure 2.2, one can see the one-hot vectors $x_{1,k}, x_{2,k}, \ldots, x_{C,k}$, representing the context of the current word in the input layer. In this model, the output layer consists of the one-hot vector of the word, $y_j$. On the other hand, Figure 2.3 displays the one-hot vector of the word $x_k$ in the input layer, while the output layer contains the one-hot vectors of the words in its context, $y_{1,j}, y_{2,j}, \ldots, y_{C,j}$. In both these models, the user can choose the dimension of the hidden layer $N$ and the size of the context $C$. After the model is trained, it is used to map each of the words in the corpus to a vector. This vector is obtained from the hidden layer of the network and it represents the word's relationship with other words. The more the two words $w_i$, $w_j$ appear in the same context, the larger the *cosine*

Figure 2.3: Skip-gram model (Source: [52])



*similarity* of their vectors vec($w_i$), vec($w_j$) is. Given two vectors $x$ and $y$, the cosine similarity is calculated as follows:

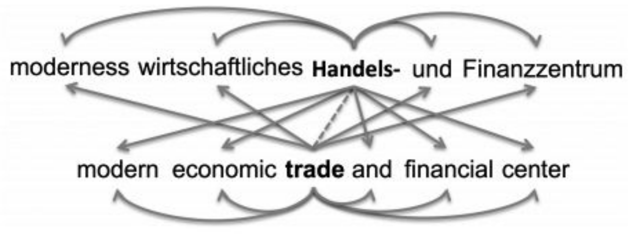
$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \times \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum\limits_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i}{\sqrt{\sum\limits_{i=1}^{n} \mathbf{x}_i^2}\sqrt{\sum\limits_{i=1}^{n} \mathbf{y}_i^2}}.$$

Word vectors obtained by training word2vec models have several interesting features. The relationship among the words are projected onto their vectors to such an extent, that for example the most similar vector to the resulting vector of an operation vec(“king”) − vec(“man”) + vec(“woman”) is the vector vec(“queen”). We will not pay further attention to more details about the advanced features of the word2vec results as they are not used in our method. It is worth mentioning that there are other related tools to word2vec, like for example GloVe [53].

The authors of bivec proposed an extension of the original Skip-gram model in the form of a joint bilingual model called the *Bilingual Skip-gram (BiSkip)* model. When trained, this model can predict the context of a given word in both languages. In order to train, bivec requires a sentence-aligned parallel corpus and its word alignment. In fact, the word alignment is not strictly necessary, and if it is not provided, the system uses a simple heuristic. However, with the alignment provided, the results are better.

The most important idea behind the BiSkip model is the method of training. To gain insight into how the training works, let us look at the example illustrated in Figure 2.4. Suppose that we have two parallel sentences with their word alignment that says the English word "trade" is aligned with the German word "Handels-". We can use the associated German word "Handels-" to predict the surrounding English context containing the words like "economic" and "financial". Generally speaking, given a word $w_1$ in a language $l_1$, and a word $w_2$ in a language $l_2$ aligned to $w_1$, the BiSkip model uses the word $w_1$ to predict not only its own context in $l_1$, but also the context of the word $w_2$ in $l_2$. The other word $w_2$ is handled analogously. This results in training of a single Skip-gram model with a joint vocabulary on parallel corpora. In other words, BiSkip model is like a single Skip-gram model trained jointly to predict words for each of the language pairs $l_1 \rightarrow l_1$, $l_1 \rightarrow l_2$, $l_2 \rightarrow l_1$, and $l_2 \rightarrow l_2$ simultaneously. The resulting vector representations have the same properties, except that in this case they appear not only monolingually but also cross-lingually. It means, for example, that given a big enough English–German parallel corpus, the resulting vector representations of the words "car" and "Auto" would have greater cosine similarity.

Figure 2.4: Bilingual Skip-gram model (Source: [51])


moderness wirtschaftliches Handels- und Finanzzentrum
modern economic trade and financial center

Similarly to word2vec being not the only one among the monolingual word embedding strategies, bivec also has some interesting relatives. We could mention BilBOWA [54] (Bilingual Bag-of-Words without Alignments), a simple and computationally efficient model, which also learns bilingual distributed representations of words. The method does not require word alignment; instead, it trains directly on monolingual data using extracted bilingual signal from a potentially smaller sentence-aligned parallel corpus.

We decided to use bivec for learning the bilingual word vectors from the training parallel corpus. The results conducted by the authors suggest that bivec achieved state-of-the-art results among other similar approaches when applied in the field of cross-lingual document classification. Using the trained bilingual word vectors and the proper weighting scheme, our method calculates the aggregate document vectors. The similarities among the word vectors are reflected in the document vectors; therefore, a pair of parallel documents have similar vectors.

## 2.6 Annoy

Annoy [55] is a C++ library with Python bindings implementing the *approximate-nearest-neighbors (ANN)* search. The *nearest-neighbor (NN)* search is an optimization problem defined as follows: given a set of points $S$ in a $d$-dimensional space $M$ and a query point $q \in M$, find the closest point in $S$ to $q$. A generalization of this problem is the *k-nearest-neighbors (k-NN)* search, where we want to find the $k$ closest points. The NN and k-NN searches require the results to be optimal. Even the best algorithms to solve these problems are computationally demanding in terms of time. However, certain applications do not require the optimal solution to be found; they just need a result that would be "good enough". In ANN search, the algorithm does not guarantee to find the optimum, but in many cases it actually does. This relaxation enables the ANN algorithms to use less resources than the k-NN strategies would need. In particular, it takes less time to find the approximate neighbors, which is what we need in our method.

There are multiple libraries implementing the ANN search. When deciding which one to choose, we were inspired by the results of the available benchmarks [56]. Annoy is fast and it is also user-friendly. It was developed during the Hack Week event at the company called Spotify[2], where it is being used to create an index for billions of feature vectors representing individual songs. This index is distributed and searched in parallel to provide music recommendations for the users.

Annoy has several useful features. First of all, the user can choose between two different metrics: Euclidean distance and angular distance. The latter one is explained by the authors to actually mean Euclidean distance of normalized vectors. The testing results show that Annoy works better with less than 100 dimensions, but the results are still efficient with less than $1,000$ dimensions. The library decouples index creation from lookup. The index needs to be built on a single machine and cannot be altered afterwards. After adding new items, the whole index needs to be rebuilt. The library allows storing the index to a file and also loading it from a file into memory. This feature enables building the index once and sharing it across all nodes in a cluster. This way, the index can be used in parallel execution. At this moment, we are not using the library in parallel execution; however, this feature may help to improve our system in the future.

In the background, Annoy uses random projections [6][7] to build up a forest of search trees—an index structure for the searching process. The algorithm for building the index uses SimHash, which is a locality-sensitive hashing (LSH) method. LSH is a family of hashing algorithms that perform dimensionality reduction of high-dimensional data. These algorithms hash items in such a way that similar ones end up close to each other, i.e. in the same buckets, with relatively high probability.

[2]`https://www.spotify.com` (accessed April 11, 2016)

We use Annoy in our method to approximate the search fo parallel documents. We already know that a pair of parallel documents have similar document vectors. Therefore, in order to search for the parallel document we can perform the nearest-neighbour search in the set of document vectors associated with the other language. In our opinion, the task of refining parallel data from the web is not the kind of task that would require an algorithm to provide a perfect recall. When it comes to processing web-scale amounts of data, one must accept certain trade-offs. We believe that by preferring ANN to k-NN we can achieve an acceptable speed of the process and, eventually, more desirable results.

## 2.7 PyBrain

PyBrain [57] (Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library) is a popular machine learning library for Python. It provides a flexible, easy-to-use, yet powerful algorithms for common machine learning tasks.

There is a part of our method, where we need to perform a binary classification task: whether to accept the pair of documents as parallel or not. As there are multiple features available for the task, it is hard to set up reasonable thresholds manually. Therefore we decided to use the classification provided by the feed-forward neural networks [58]. For this purpose we use the neural network algorithms provided by the PyBrain library.

The model of neural network used in our method is multilayer perceptron learning through the mechanism of backwards error propagation (backpropagation) [59]. This kind of model is widely popular and it is suitable for the type of classification task the method is facing.

## 2.8 Common Crawl

In Section 1.5, we have discussed the previous work concerning refining parallel corpora from Common Crawl dataset. The following text provides additional information about the dataset. The Common Crawl Foundation [11] is a non-profit organization with an aim of democratizing access to web information. They produce and maintain an open repository containing web-crawled data freely available for the public to use.

The underlying infrastructure of Common Crawl uses Apache Nutch[3] as the primary web-crawling engine since 2013. Nutch runs in the form of Hadoop MapReduce jobs, which perform most of the core work of fetching pages, filtering and normalizing URLs and parsing responses to plug-ins. The plug-in architecture

[3]`http://nutch.apache.org/` (accessed April 7, 2016)

of Nutch allows the Common Crawl organization to make all the customizations they need, without maintaining a separate branch of the Nutch project. The infrastructure regularly crawls at aggregate speed of $40,000$ web pages per second. The performance is largely limited by the politeness policy, which the organization follows to minimize the impact on web servers the system crawls.

The time to release a new dataset (crawl) varies between one month to a quarter of year. Each crawl covers billions of web pages and takes hundreds of terabytes (TB) of disk space in an uncompressed form. Table 2.2 lists few of the crawls, along with their uncompressed size and number of contained pages.

Table 2.2: Common Crawl dataset sizes

| **Crawl** | **Size (TB)** | **Pages (Billions)** |
|---|---:|---:|
| April 2015 | 168 | 2.11 |
| May 2015 | 159 | 2.05 |
| June 2015 | 131 | 1.67 |
| July 2015 | 149 | 1.84 |
| August 2015 | 106 | 1.32 |

The crawls are hosted by the Amazon Web Services[4] as a part of the Public Data Sets[5]. They are stored in a Simple Storage Service (S3)[6]. The service allows the user to download the whole crawls to a local cluster. There is also a possibility to work with the datasets using the Amazon services, but these are paid. The Common Crawl foundation provides the users with the latest crawls in 3 different data formats:

- The *WARC* [60] (Web ARChive) format represents the raw archive of the crawl. It is a container for storing web content together with the associated network connections. It consists of series of records. There are three types of records: HTTP request, HTTP response and metadata describing the crawl process. Unlike the other two, it keeps the original HTML structure of the responses.
- The *WAT* (Web Archive Transformation) format contains metadata extracted from the WARC records formatted in JSON [61]. These provide information such as content length, timestamp, mime type and URL.
- The *WET* (WARC Encapsulated Text) format includes extracted plain text from the archived documents in WARC along with some basic metadata such as timestamp and URL.

Our other experiment uses realistic and noisy data from the July 2015 crawl. For the experiment, we have downloaded all the WARC files to our cluster, to

[4]`https://aws.amazon.com/` (accessed April 4, 2016)
[5]`https://aws.amazon.com/public-data-sets/` (accessed April 8, 2016)
[6]`https://aws.amazon.com/s3/` (accessed April 8, 2016)

be processed in our environment. We use the WARC format, because unlike the WET format it holds also the original HTML structure for the stored web pages. Using the HTML structure, we are able to extract paragraphs reliably. We consider these paragraphs as an input documents to be aligned.

## 2.9 Hadoop

Common Crawl datasets belong to the category of Big Data [39]. When processing hundreds of terabytes large datasets, one definitely needs a more complex infrastructure and parallel processing paradigm. Apache Hadoop[7] [62] is an open-source Java-based [63][64] software framework, which provides distributed storage along with means for distributed processing. It is designed to process large datasets on computer clusters built from commodity hardware. Hadoop is based on ideas originating in Google that invented an infrastructure able to process the data from the web at a large scale. The original distributed storage was called the Google File System [65] and the processing framework was named MapReduce [37]. In the beginning, Hadoop was developed by employees of Yahoo! company as a part of the Apache Nutch project. Later it was separated to a standalone project.

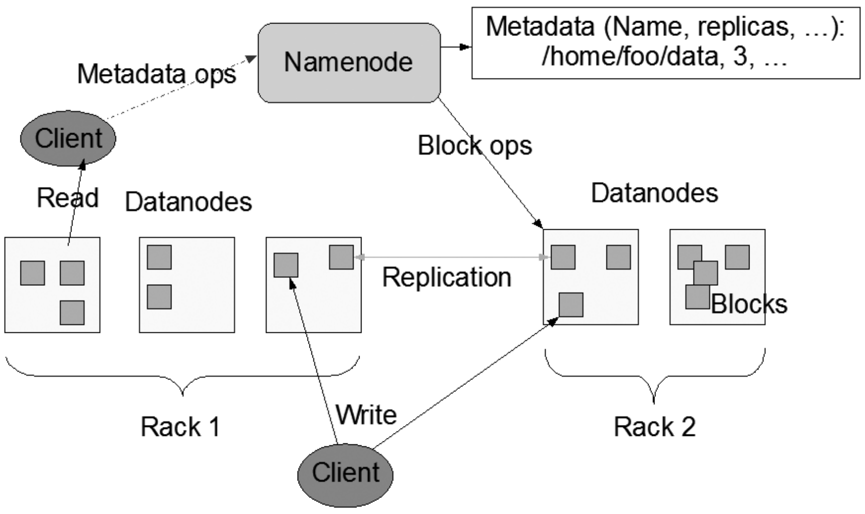
Today, Hadoop is composed of many different components working together on a single platform. Historically, there are two main components in Hadoop. The storage for the data called HDFS (Hadoop Distributed File System) [66] and computational framework for parallel executions—MapReduce.

### 2.9.1 HDFS

HDFS [67] is a scalable storage, which enables user to store large files to a cluster in a distributed manner. It has a master/slave architecture illustrated in Figure 2.5. An HDFS cluster has a master server, called NameNode. It manages the file system namespace and regulates file access rights. Moreover, there is a number of slaves, named DataNodes, present within the cluster, usually one per node. Each of the DataNodes manages storage on its node. HDFS exposes the file system namespace allowing the user to store the data. The internal implementation splits a file into a group of blocks that are distributed across a set of DataNodes. This allows huge files to be stored. The NameNode determines the mapping of blocks to DataNodes. It is also responsible for operations like opening, closing and renaming of files and directories. The DataNodes serve read and write requests from clients. They also provide an interface for block creation, deletion and replication to the main NameNode. The mechanism of data replication is centrally managed by the NameNode.

[7] `http://hadoop.apache.org/` (accessed April 9, 2016)

Figure 2.5: HDFS architecture (Source: [67])



## 2.9.2 MapReduce

MapReduce [68] is a software framework, which allows to process vast amounts of data on large clusters. It is reliable and fault-tolerant. A MapReduce job splits the input data into independent chunks. These are first processed in parallel by the map tasks (mappers). With map tasks finished, the framework sorts their output and the results are passed to the reduce tasks (reducers). The framework is responsible for scheduling, monitoring and re-execution of the tasks. In a typical scenario, the MapReduce framework and HDFS are running on the same set of nodes. This configuration allows the framework to effectively schedule the tasks to nodes where the data reside. This way, the system minimizes the amount of network traffic needed for job execution. Similarly to HDFS, also MapReduce has a master/slave architecture. The framework consists of a single master ResourceManager, multiple NodeManagers (one per node of a cluster), and one MRAppMaster per application. The framework application has to specify the input and output locations and supply the implementations for the map and reduce tasks. These and other settings form a job configuration. The process starts when a job client submits a job with a JAR file (Java Archive) or executable to the ResourceManager. The ResourceManager then distributes the software and configuration to the slaves and begins scheduling the tasks. During the execution, it provides status and diagnostic information.

The framework operates exclusively using *key-value* pairs. It is a fundamental data structure providing users with extensibility. In a key-value based data model the information is stored within collections in which every item is a pair of key and value. Within the framework, the input of a job is a set of key-value pairs. The output of the job is another set of key-value pairs of possibly different types. The classes of keys and values need to be serializable by implementing a dedicated interface. Additionally, the key classes need to be comparable. The standard dataflow of the MapReduce job execution is following:

1. The system first divides the input data into splits. These splits have usually 64–128 megabytes (MB). The framework assigns one split to each mapper. Then it begins to read the input data from the HDFS, generating key-value pairs. These pairs are passed to the mappers. In a typical case, if processing plain text files, a key-value pair would be generated for each line in every file located in an input directory.

2. The mapper gets a series of key-value pairs. While processing one input pair it can generate zero or multiple output key-value pairs. The types of input and output keys and values may differ.

   Listing 2.1 shows an pseudo-code implementation of a map function from the classical WordCount example which performs extraction of word frequencies. In this case the function is called for every line contained in the assigned split. The mapper gets the line and divides it into its words. Then for each of these words it emits an output key-value pair, where key is the word and value is 1. The value represents a single occurrence of the word.

3. Every output of each mapper is assigned to a specific reducer. The partition function gets the mapper output key and the actual number of reducers and it determines the associated reducer. Then the data are shuffled—they are sorted in parallel, using the provided comparison function, and exchanged between the nodes of mappers and reducers. The performance of this phase is dependent on the speed of the network between cluster nodes.

4. For each unique key, in sorted order, a reducer is called. Reducer iterates over all the values associated with the given unique key. It can generate zero or multiple output key-value pairs. Like in mapper, also in reducer the types of input and output keys or values may differ.

   The implementation of a reduce function from the word count example is presented in Listing 2.2. It sums all the partial counts emitted by mappers into a total count for each of the unique words.

5. Finally, the system stores the output of the reducers to the HDFS. The framework allows the user to define MapReduce batch jobs for executing multiple jobs in a sequence.

Listing 2.1: WordCount: map

```
function map(String name, String line):
    for word in line.split():
        emit(word, 1)
```

In order to store and process the July 2015 dataset of the Common Crawl (149 TB, 1.84 billions of pages), we needed a distributed storage and framework for parallel data processing. Therefore, we decided to use the Apache Hadoop for these purposes. When wondering about other means of processing of such a huge dataset, one can get inspired by the list of example projects using Common

Listing 2.2: WordCount: reduce

```
function reduce(String word, Iterator partial_counts):
    total_count = 0
    for count in partial_count:
        total_count += count
    emit(word, total_count)
```

Crawl data[8]. One of the examples mentioned in the list is the method described in Chapter 1.

## 2.10 WARC-Hadoop

When processing data in a custom format (e.g. WARC file format) with Hadoop, the user needs to implement a set of interfaces for the system to handle the data properly. There exist a number of libraries which enable Hadoop to process the WARC files with the MapReduce framework. We decided to utilize one such library, called warc-hadoop [69] in our experiment.

The warc-hadoop is a Java library providing the functionality for reading and writing WARC files in MapReduce environment. It also enables the framework to serialize and transfer individual WARC records while shuffling the data between the mappers and reducers. This library was created with an intention to help the users explore the content of the Common Crawl datasets.

## 2.11 MetaCentrum

The Hadoop cluster we use for our experiments is provided by the MetaCentrum[9] project, which is an activity of the CESNET[10] association. MetaCentrum operates and manages distributed computing infrastructure, which consist of computing and storage resources owned by CESNET and cooperating academic centers located in the Czech Republic. Moreover, MetaCentrum holds responsibility for building the National Grid and its integration with related international activities.

The provided Hadoop cluster in MetaCentrum consists of 3 management nodes and 24 worker nodes. The management nodes run components like front-end, HDFS NameNode and MapReduce History Server. Every node of the configuration has Intel® Xeon® CPU E5-2630 v3 (20 MB Cache, 2.40 GHz) and 128

[8] `http://commoncrawl.org/the-data/examples/` (accessed April 10, 2016)
[9] `https://www.metacentrum.cz/en/` (accessed April 10, 2016)
[10] `https://www.cesnet.cz/?lang=en` (accessed April 10, 2016)

gigabytes (GB) of memory. The overall disk space available on the cluster is 1.02 petabytes (PB). The HDFS operates with a replication factor of 4, which means that the cluster can hold de facto 261 terabytes (TB) of user data.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme ”Projects of Large Research, Development, and Innovations Infrastructures” (CESNET LM2015042), is greatly appreciated.

## 2.12 jsoup

Processing data from the web is usually complicated. For example, there are many web pages, that do not have valid HTML structure or are encoded in different character sets. To tackle this we use a library called jsoup [70], a Java library for working with real-world HTML structures. It transforms the HTML structure to the Document Object Model (DOM) [71].

The jsoup library provides a way to extract data using DOM traversal or Cascading Style Sheets (CSS) selectors. Furthermore it allows the user to clean the HTML structure by keeping only the HTML elements and attributes present in a provided whitelist.

In our real-world data experiment, we process Common Crawl dataset with the native Java MapReduce library. Therefore we consider jsoup as suitable library for our needs. We use the library in parallel execution environment to parse the HTML structures of the web pages. The library allows us to scrape the contents of the HTML elements `<p>` representing paragraphs of text that we consider as input documents to be aligned by our method. We could also collect content of other elements such as `<h1>`–`<h6>`, `<li>`, `<th>`, `<td>`, etc., as well.

## 2.13 language-detector

Another Java library we use in our real-world data experiment is language-detector [72]. As the name suggests, it is a library that performs language detection. It contains built-in profiles for 70 different languages, which were trained using common texts available for each of the languages. The library also allows the user to train a custom language profile when provided with a monolingual corpus. Underneath, the library uses N-gram-based text categorization [73].

We use this library while running MapReduce over the Common Crawl dataset. For each of the paragraphs scraped from the web pages, we identify the language using the language-detector library. This way the MapReduce job is able to keep only the paragraphs in the languages of our interest.

# Chapter 3

# Proposed Method

This chapter describes our solution to the task of bilingual document alignment. The method has been already outlined in Section 2.1 of Chapter 2 containing an introduction to all the resources and tools used in our work. The following text discusses our method in a greater level of detail.

## 3.1 Task Definition

Our method solves an extended version of the bilingual document alignment task. This version of the task can be defined as follows: let us assume we have a number of sets containing documents from both the languages of our interest. We call these sets *bins*. Each bin represents a stand-alone set of input documents for the original task. The solution of this task is a collection of pairs of documents where each pair consists of two parallel documents from the same bin.

A bin can contain up to millions of documents and is not required to have a balanced language distribution. Individual bins may vary in size. The method does not align the documents either belonging to the different bins or having the same language. Intuitively, the smaller the size of bin, the better the quality of the resulting alignment. It also takes more time and memory to align a larger bin.

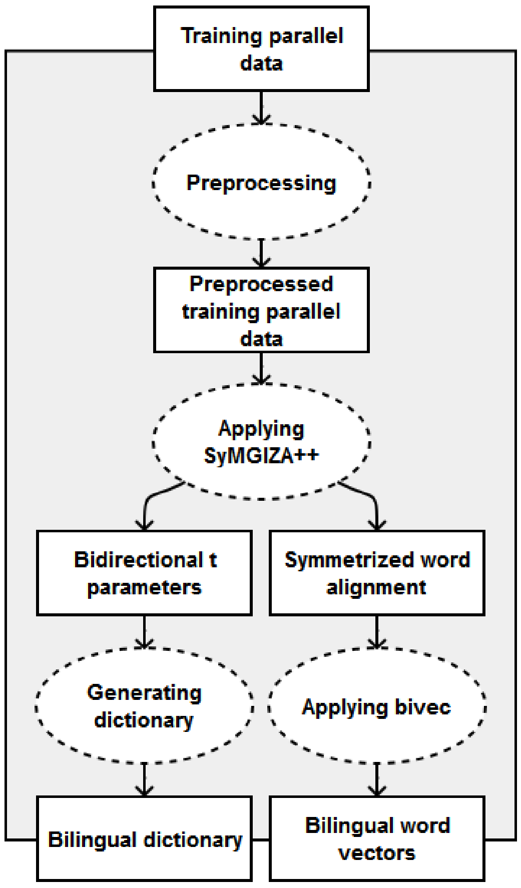
When mining bilingual parallel corpora from the web, one can form a bin for every identified bilingual web domain. Such a bin can contain all the paragraphs in both the languages, scraped from the domain. This way the method will consider aligning of all the paragraphs from the domain regardless of the URLs and HTML structures of its web pages.

## 3.2 Training Part I: Dictionary, Word Vectors

Our method is supervised and needs to be trained on an already existing sentence-aligned corpus for the language pair we are interested in. For better clarity, we distinguish between two parts of the training process. This section describes the first part of training, while the other part is described in Section 3.3. When trained, Section 3.4 explains how the method aligns the input data.

Let us describe the first part of the training process as depicted in Figure 3.1. Within the diagram, rectangles represent data in various formats, ovals stand for processing steps and arrows symbolize the flow of the process. The objective of this part of the training, is to preprocess the training parallel corpus and create a bilingual dictionary together with bilingual word vectors. The process is described in the following text.

Figure 3.1: Proposed method: training part I



### 3.2.1 Preprocessing Training Parallel Data

The procedure begins with preprocessing of the sentence-aligned training parallel corpus which may involve tokenization, lemmatization, stemming, truecasing, lowercasing, removing unknown symbols, removing stop words, etc. For individual language pairs, different preprocessing steps might help to gain better quality results. Any type of preprocessing done in this step needs be also applied to the input data, before the alignment process starts, for the method to work properly. However, this does not mean, that the user must end up with pairs of parallel documents in preprocessed format. The method is easily extensible to be able to track down the original documents.

In our experiments with Czech–English language pair, the preprocessing includes either tokenization or lemmatization, followed by lowercasing. The tokenization and lemmatization is done utilizing MorphoDiTa (see Section 2.3), an open-source morphological dictionary and tagger.

### 3.2.2 Applying SyMGIZA++

The method follows the well-known recommendations to get a good-quality word alignment. The resulting corpus from the previous step is further cleaned by removing all the sentence pairs, where one of the sentences contains more than 50 tokens or does not contain a single letter from any alphabet.

Then SyMGIZA++ (see Section 2.4) is executed to obtain the word alignment for the preprocessed and cleaned training parallel corpus. This step includes preparation of word classes and word co-occurrences which are used in the alignment process. The results of the execution include the values of the IBM Model 1 "t" parameters, after its last iteration, for both directions.

### 3.2.3 Generating Dictionary

The bilingual dictionary is built using the final IBM Model "t" parameters estimated by SyMGIZA++. The algorithm is the same as the one implemented in Bitextor's script for creating a custom dictionary (see Section 1.1). Each word pair that appears in both directions and has the harmonic mean of the "t" parameters (i.e. *weight*) great enough, is included into the dictionary. Unlike the Bitextor's one, this type of a dictionary includes also the weights.

We created a script `merge_param.py` which takes the SyMGIZA++ vocabulary files for both languages, files containing "t" parameters for both directions, a threshold for the weights and produces a bilingual dictionary. Listing 3.1 shows a sample from the file containing bilingual dictionary created by this script.

By default, we keep all the pairs of words with weight more than 0.00001. The threshold is set relatively low, producing large dictionaries. Searching through a larger dictionary takes more memory and computational time which heavily affects the overall performance of the method. The reasoning behind such a low threshold is that we wanted to demonstrate the limits of our method's preci-

Listing 3.1: Sample from a file with bilingual dictionary (training)

```
řekl    said    0.448062
řekl    told    0.162753
řekl    say     0.0364408
řekl    tell    0.0109902
```

sion rather than its performance. Yet, we consider the chosen threshold as still acceptable when talking about the resource requirements of the method.

### 3.2.4 Applying bivec

The input files for bivec (see Section 2.5) training are obtained by vertically splitting SymGIZA++ output file `all.A3.final_symal`. Each line of this file contains a pair of parallel sentences along with their word alignment in a format introduced by Pharaoh[1], a machine translation decoder.

It is worth noting that bivec was built to accept alignment format of Berkeley Aligner[2], which is another word alignment tool. To transform the Pharaoh format into the required one, all the dashes must be replaced with spaces.

The method follows the recommendations of how should be the training data preprocessed. All the sequences of numbers are replaced with a zero symbol and all the unknown symbols (e.g. non-printable Unicode characters) with the specially dedicated tag `<unk>`.

With all the input files for training prepared, bivec is executed to create the bilingual word vectors. It is set to generate vectors with 40 dimensions. Listing 3.2 and Listing 3.3 show samples taken from the files containing bilingual word vectors produced by bivec. Additionally, Table 3.1 lists a sample of cosine similarities between the word vectors. In the table, the order of the Czech and English words is the same, so the diagonal represents translations.

There is a reason, why we keep the number of dimensions relatively low. The word vectors are used to calculate the aggregate document vectors with the same number of dimensions. The document vectors are then indexed using Annoy (see Section 2.6). The authors of Annoy suggest that the tool works best with number of dimensions less than 100. On the other hand, the authors of bivec conducted the tests using 40, 128, 256 and 512 dimensions. We have decided to use the only number of dimensions suitable for Annoy that has been tested.

[1]`http://www.isi.edu/licensed-sw/pharaoh/` (accessed May 2, 2016)
[2]`https://code.google.com/archive/p/berkeleyaligner/` (accessed April 13, 2016)

Listing 3.2: Sample from a file with Czech word vectors (training)

```
řekl 0.610664 0.186801 0.586637 -0.305300 0.785947 -0.114462 -0.168189
    -0.800271 0.761297 -0.286534 0.195719 -0.125131 -0.821144 0.049325
    -0.603093 -0.183007 0.240985 0.083267 0.144988 -0.375526 0.269821
    -0.266884 0.141238 0.163624 -0.385829 0.255967 -0.700835 0.451331
    0.341263 0.333853 0.177087 -0.085332 -0.222975 0.753013 0.005252
    0.023802 -0.520247 -0.062342 -0.485972 -0.216207
```

Listing 3.3: Sample from a file with English word vectors (training)

```
said 0.601102 0.260525 0.566347 -0.263702 0.673600 -0.114424 -0.137723
    -0.704463 0.619913 -0.402364 -0.043697 -0.052677 -0.862785 0.107025
    -0.665232 -0.119659 0.101142 -0.086549 -0.105953 -0.572788 0.379709
    -0.309156 0.056748 0.016574 -0.131031 0.380851 -0.356606 0.340167
    0.374560 0.466035 0.319632 -0.070731 -0.221821 0.630211 0.117143
    0.033079 -0.416265 -0.012100 -0.469880 -0.166465
```

Table 3.1: Sample of cosine similarities between word vectors

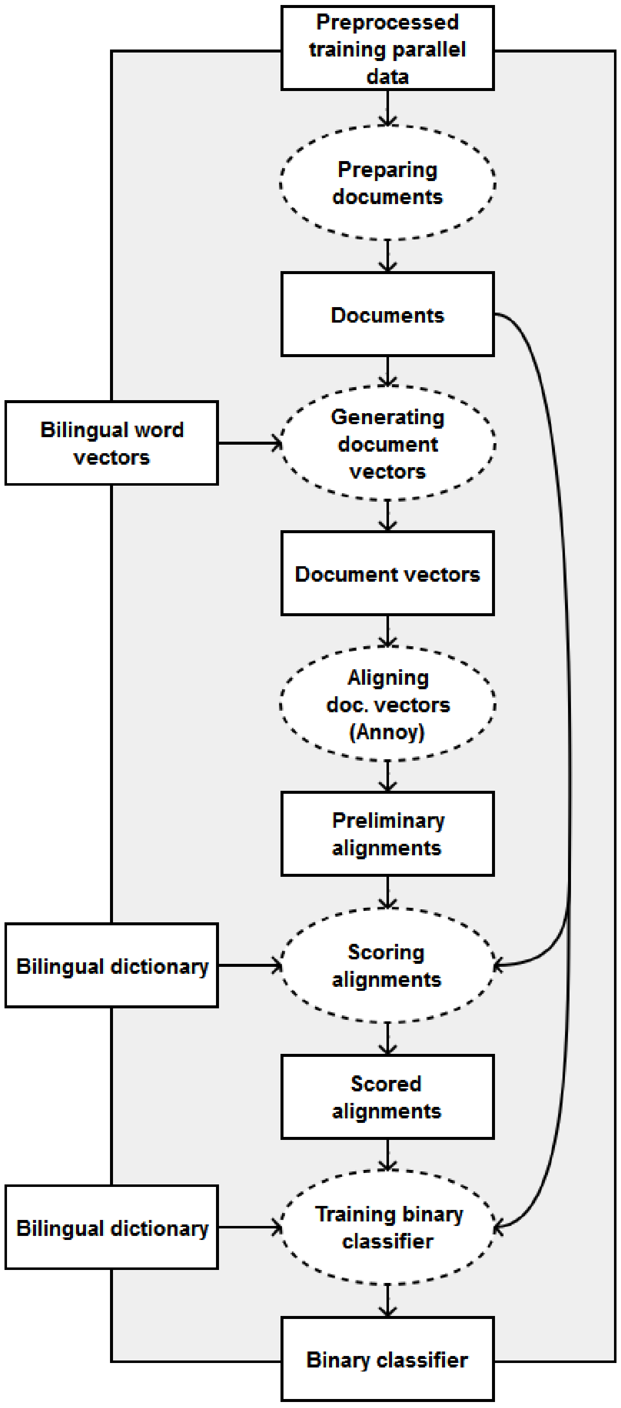
| | **řekl** | **kočka** | **pes** | **káva** | **čaj** |
|---|---|---|---|---|---|
| **said** | **0.9522** | 0.4480 | 0.4403 | 0.4492 | 0.5440 |
| **cat** | 0.4070 | **0.9383** | **0.8350** | 0.5053 | 0.5346 |
| **dog** | 0.4950 | **0.7727** | **0.9328** | 0.3900 | 0.5582 |
| **coffee** | 0.4682 | 0.4423 | 0.3880 | **0.8456** | **0.9041** |
| **tea** | 0.5014 | 0.4202 | 0.4232 | **0.8129** | **0.9698** |

## 3.3 Training Part II: Classifier

With the first part of the training done, the method has prepared the preprocessed training parallel corpus (see Section 3.2.1), bilingual dictionary with weights (see Section 3.2.3), and bilingual word vectors (see Section 3.3.2).

The second part of the training process is illustrated in Figure 3.2. The process is almost the same as the procedure of running the trained method. The difference is that in training, we are aligning a supervised dataset with the intention to train a binary classifier able to decide whether to accept a pair of documents as parallel or not. The trained classifier is then used when running the trained method on the input data. The following text describes the procedure of the second part of the training.

Figure 3.2: Proposed method: training part II



### 3.3.1 Preparing Documents

The first step creates the supervised dataset. The dataset consists of bins containing bilingual parallel documents. It is formed from the pairs of parallel sentences contained in the preprocessed training parallel corpus (see Section 3.2.1). These are considered as pairs of parallel documents for the training.

In Section 3.1, we have explained the concept of the bins. The method splits all the pairs of documents into equally large bins (except the last one). The size of the bin should be an estimate of an expected size of the bin in a real-world dataset. In our experiment we split the documents into training bin consisting of $50,000$ pairs of parallel documents, i.e. $100,000$ documents. We consider this value as an upper-bound estimate of an average number of paragraphs in either of the two languages located on an ordinary bilingual web domain.

Our implementation accepts the supervised dataset in the form of two files, one for each language. Listing 3.4 and Listing 3.5 show samples of such a pair of files. The format is a list of tab-separated values. The first column is an identifier of the bin. The samples show those parts of the files, where the first bin ends and the second bin starts. For the implementation to be able to iterate over both these files simultaneously, it is required that the bins are sorted in an alphabetic order. The second column is an identifier of the document. This identifier needs to be unique at least within the bin. The third column it the textual content of the document. The examples show documents from the tokenized and lowercased training parallel corpus. For the training to be successful, it is essential that each pair of parallel documents share the same bin and document identifier. This way the method knows which pairs of the documents are parallel.

Listing 3.4: Sample from a file with Czech documents (training)

```
00000000    49998   " stál jsem támhle u zdi , " řekl .
00000000    49999   " nešpehoval jsem , harry .
00000001    50000   nikam nechodíme .
00000001    50001   pokračujte pane farbere .
```

Listing 3.5: Sample from a file with English documents (training)

```
00000000    49998   ' i was standing there by the wall , ' he said .
00000000    49999   ' i was n't spying , harry .
00000001    50000   we never socialize .
00000001    50001   continue , mr . farber .
```

## 3.3.2 Generating Document Vectors

For each document, an associated vector is generated using the bilingual word vectors obtained in the first part of the training (see Section 3.2.4) together with the *tf-idf* (term frequency-inverse document frequency) weighting scheme. The tf-idf weight of a term $d_i$ in a document $d = (d_1, d_2, \ldots, d_n)$ can be expressed as:

$$\text{tf-idf}(d_i, d) = \text{tf}(d_i, d) \times \text{idf}(t) = \text{tf}(d_i, d) \cdot \log\left(\frac{N}{\text{df}(d_i)}\right)$$

where $\text{tf}(d_i, d)$ is the number of occurrences of the term $d_i$ in the document $d$, $\text{df}(d_i)$ is the number of documents containing the term $d_i$, and $N$ is the number of all the documents in a dataset.

Our implementation processes both the files with documents one by one, bin by bin. When processing a bin, all the duplicate documents present in the bin are first discarded. Then, the inverse document frequencies are calculated for all the words that appear in the bin. Lastly, the document vector for every unique document $d = (d_1, d_2, \ldots, d_n)$ is generated as:

$$\text{docvec}(d) = \sum_{i=1}^{n} \text{tf-idf}(d_i, d) \times \text{wordvec}(d_i)$$

where $\text{wordvec}(d_i)$ is the word vector for the term $d_i$. The product operation in the formula is a multiplication of a scalar with a vector and the summation is derived from the operation of vector addition. If a word vector does not exist for a given term, a zero vector is used instead.

The described procedure is implemented in a script called `create_docvec.py`. When given a file with documents and a file with word vectors for the associated language, it generates an output file containing document vectors. Listing 3.6 and Listing 3.7 show a sample of its output. These lines contain the document vectors produced for the first pair of parallel documents displayed in Listing 3.4 and Listing 3.5. The output format of the script is very similar to the format of a file with documents. The only difference is that the document contents are replaced with document vectors.

Listing 3.6: Sample from a file with Czech document vectors (training)

```
00000000    49998   8.180257 -6.041753 6.456024 -7.385942 4.504289
    -2.480280 -2.110008 -6.952853 9.294062 -5.956102 0.873277 0.288546
    -8.055108 3.530353 -13.852273 1.189734 6.368119 4.307136 2.640194
    -5.734687 -3.508690 -3.307812 -4.178317 -5.088661 -2.772588
    10.505361 -7.485562 5.391955 5.723570 6.392571 3.516147 -1.386106
    -5.184054 11.635170 -7.812555 6.185200 -0.854625 2.147744 -5.315508
    1.234217
```

Listing 3.7: Sample from a file With English document vectors (training)

```
00000000    49998   3.407737 -6.297829 7.404373 -6.480918 3.653056
    0.363053 -0.486428 -4.324950 7.306768 -3.102215 -0.762884 -1.544800
    -6.174080 0.820872 -10.864064 -1.982250 4.873059 -1.361170 -2.714015
     -4.592477 2.429560 -1.330689 -4.640379 -3.568162 -1.444272
    10.333115 -5.498119 1.396311 3.515478 9.095764 2.544416 -1.068039
    -5.637783 5.594423 -3.433248 4.237183 0.336851 -0.263934 -2.851157
    2.225453
```

### 3.3.3 Aligning Document Vectors (Annoy)

For each bin, the following procedure is performed. First, a search index is built containing the vectors of all the bins' documents in *target language*. To build the search index, the method uses Annoy (see Section 2.6) set to operate with the angular distance. Then, for every bin's document in the *source language*, the index is searched to obtain $k$-approximate-nearest-neighbours to its vector. This returns a list of candidate parallel documents in the target language to the document in the source language. We call these preliminary alignments.

This procedure is implemented in a script called `align_docvec.py`. When given files with the document vectors for both the languages, it creates an output file containing preliminary alignments. Listing 3.8 shows a sample of its output. The first column is the bin identifier. The second and the third columns represent the identifiers of the documents in the source and the target language, respectively. The last column contains the similarity derived from the distance provided by Annoy calculated as $1-(d/2)$, where $d$ is the returned angular distance explained to be actually a Euclidean distance of normalized vectors. In the case presented in the sample, the parallel document ended as the 9th best candidate (see line 4573319 in Listing 3.8).

Listing 3.8: Sample from a file with preliminary alignments (training)

```
457311 00000000    49998   44560   0.800532951951
457312 00000000    49998   11723   0.791310846806
457313 00000000    49998   9227    0.787315234542
457314 00000000    49998   33875   0.781678438187
457315 00000000    49998   18861   0.779217585921
457316 00000000    49998   24646   0.771993637085
457317 00000000    49998   9232    0.771212115884
457318 00000000    49998   48420   0.770708605647
457319 00000000    49998   49998   0.770486533642
457320 00000000    49998   20284   0.768467396498
```

### 3.3.4 Scoring Alignments

Within the preliminary alignments, the top candidates are not necessarily the optimal ones. Therefore, the method applies a scoring function to reorder the candidates. This increases the probability of the optimal documents to appear higher in their candidate lists. Given the document $d = (d_1, d_2, \ldots, d_n)$ and its candidate $c = (c_1, c_2, \ldots, c_m)$, the scoring function is defined as:

$$\text{score}(d, c) = \text{length\_sim}(d, c) \times \text{weight\_sim}(d, c)$$

Both the functions $\text{length\_sim}(d, c)$ and $\text{weight\_sim}(d, c)$ have the range of $[0, 1]$. The idea is that the higher the result they return, the greater the possibility that the pair is parallel. These functions are defined as follows.

- $\text{length\_sim}(d, c)$ examines the ratio of the documents' lengths. It is based on the probability density function of the normal (Gaussian) distribution:

$$\text{length\_sim}(d, c) = e^{-\frac{(\frac{\text{len}(c)}{\text{len}(d)} - \mu)^2}{2\sigma^2}}$$

  where $\frac{\text{len}(c)}{\text{len}(d)}$ is the actual ratio of the documents' lengths and $\mu$ is the expected ratio with the standard deviation $\sigma$. The expected ratio of documents's lengths with the associated standard deviation can be estimated using the pairs of parallel sentences from the preprocessed training parallel corpus. For the tokenized Czech–English parallel corpus these values are estimated to be $\mu_{cs \to en} \approx 1.08$ and $\sigma_{cs \to en} \approx 0.28$.

- $\text{weight\_sim}(d, c)$ is based on the IBM Model 1 [19] and uses the bilingual dictionary created in the first part of the training. It is defined as:

$$\text{weight\_sim}(d, c) = \prod_{i=1}^{n} \sum_{j=1}^{m} \frac{\text{weight}(d_i, c_j)}{m}$$

  where $\text{weight}(d_i, c_j)$ is the weight of the word pair $\langle d_i, c_j \rangle$ provided by the dictionary if the entry exists, otherwise it equals $10^{-9}$ ("null weight").

A script called `score_align.py` implements the described procedure. Given a file with preliminary alignments and files with the documents for both the languages, it creates an output file containing scored alignments. Listing 3.9 shows a sample of its output. The output format is almost unchanged when compared with the format of a file with preliminary alignments. The only difference is that the similarity in the last column is replaced with the calculated score. The presented sample shows scored candidates from Listing 3.8. The matching document is now considered as the top candidate.

Listing 3.9: Sample from a file with scored alignments (training)

```
00000000    49998   49998   1.21903368318e-14
00000000    49998   20284   9.19687934061e-20
00000000    49998   11723   1.55923045256e-23
00000000    49998   9232    9.73231577325e-25
00000000    49998   18861   7.95893854924e-27
00000000    49998   48420   8.82180461894e-28
00000000    49998   33875   1.19519536122e-30
00000000    49998   9227    1.70133029025e-38
00000000    49998   44560   2.16354386116e-43
00000000    49998   24646   9.5437947187e-55
```

### 3.3.5 Training Binary Classifier

It is required from the binary classifier to be able to decide whether to accept a pair of documents as parallel or not. The chosen model for the classifier is a feed-forward neural network [58]. The method uses an implementation provided by PyBrain (see Section 2.7). The classification is based on 4 features. All of these features have the range of $[0, 1]$. Given the document $d = (d_1, d_2, \ldots, d_n)$ and its candidate $c = (c_1, c_2, \ldots, c_m)$, the following list describes all the features.

- length_sim$(d, c)$ has been already defined (see Section 3.3.4). This function scores the ratio of the documents' lengths against the expected ratio.
- length_conf$(d, c)$ provides a supplementary information for the previous feature, which is not a reliable nor effective when scoring pairs of short documents; however, it is substantial when comparing pairs of long documents:

  $$\text{length\_conf}(d, c) = 1 - e^{-0.01 \times \text{len}(d)}$$

  This is a monotonically increasing function, that provides the model with an information of absolute length of the document $d$. The name of the feature is an abbreviation of "length confidence", which is justified by the fact that the higher the value of the length_conf$(d, c)$ is, the more authoritative is the score of the length_sim$(d, c)$.
- weight_sim$_2(d_i, c_j)$ is a modified version of weight_sim$(d, c)$ (see Section 3.3.4). The original version was tested for the purposes of the classification, but the results were poor. This might be caused by the fact, that it returns very small values affected by the number of words contained in both the documents to a large extent. The modified version is defined as:

$$\text{weight\_sim}_2(d,c) = \frac{\sum_{i=1}^{n} \text{len}(d_i) \times \max_{j=1}^{m}(\text{weight}_2(d_i,c_j))}{\sum_{i=1}^{n} \text{len}(d_i) \times \text{sgn}(\max_{j=1}^{m}(\text{weight}_2(d_i,c_j)))}$$

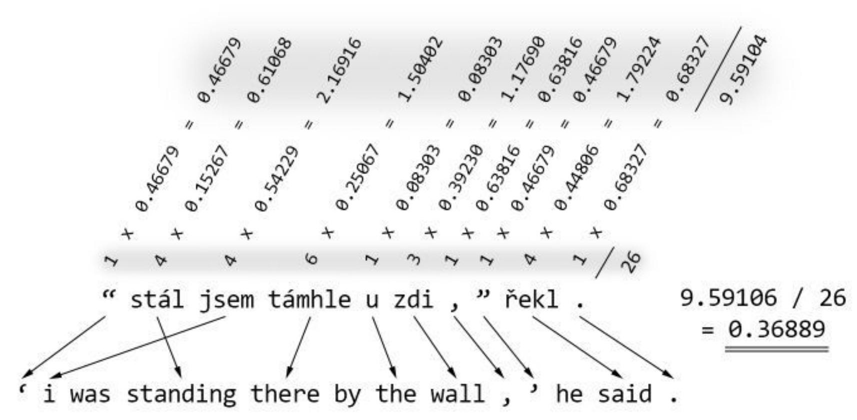
where $\text{weight}_2(d_i, c_j)$ is defined as the weight of the word pair $\langle d_i, c_j \rangle$ provided by the dictionary if the entry exists; however, if it does not exist and the two words are identical, then it equals 1, otherwise it returns 0.

Let us explain the reason behind the heuristic of $\text{weight}_2(d_i, c_j) = 1$ for a pair of identical words not having entry present in the dictionary. The same set of features is used in the running process where occurrences of new words or special terms (e.g. URLs or email addresses) are expected. The heuristic considers a pair of identical words to be a perfect translation only if the dictionary does not contain other relation.

Additionally, let us discuss why the weights are multiplied by the lengths of words. The assumption is that longer words are usually less frequent, carry more meaning and are therefore more important for the sentence, in contrast to short tokens (e.g. "," or "a"). The definition of $\text{weight\_sim}_2(d, c)$ is an arithmetic mean of strongest relations between a source word from $d$ and any of the target words from $c$, weighted by the lengths of source words.

Figure 3.3 shows an example of weight_sim$_2$ calculation. In the example, for every source word in the Czech document, there exists an entry in the bilingual language dictionary with at least one of the target words from the English candidate document. Arrows represent the strongest among the relations (max weight$_2$) for each of the source words. The calculation above each of the source words is a multiplication of the length of the source word and the weight of the associated strongest relation.

Figure 3.3: Example weight_sim$_2$ calculation



• weight_conf$_2$ is a supplementary feature for weight_sim$_2$. We can interpret the weight_sim$_2$ feature as: "From what we know with the knowledge of possible incomplete dictionary, how likely are these two documents parallel?".

The supplementary feature can be similarly interpreted as: "To what extent the dictionary covers the pairs of words we came across?". The formal definition is following:

$$\text{weight\_conf}_2(d, c) = \frac{\sum_{i=1}^{n} \text{len}(d_i) \times \text{sgn}(\max_{j=1}^{m} (\text{weight}_2(d_i, c_j)))}{\sum_{i=1}^{n} \text{len}(d_i)}$$

With all the features designed for the classification defined, the process of training can be explained. It starts by creating a training dataset using the scored alignments. For every document $d$ in the source language and its top candidate $c$ in the target language the following pair of input→output vectors is added into the training dataset:

$$\begin{pmatrix} \text{length\_sim}(d, c) \\ \text{length\_conf}(d, c) \\ \text{weight\_sim}_2(d, c) \\ \text{weight\_conf}_2(d, c) \end{pmatrix} \rightarrow \begin{cases} \begin{pmatrix} 0 \\ 1 \end{pmatrix}, & \text{if } \langle d, c \rangle \text{ are parallel} \\ \\ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & \text{otherwise.} \end{cases}$$

The input vector consists of the 4 defined features, while the output vector encodes whether the documents $\langle d, c \rangle$ are parallel or not. The first value of the output vector represents the probability of the documents to be non-parallel. The second value is complementary to the first.

Before the network is trained, the collected training dataset is subsampled to contain an approximately equal number of items representing parallel and non-parallel document pairs. This helps the network to be less affected by the ratio of parallel and non-parallel pairs. At this moment, it is also possible to reduce the size of the dataset to shorten the time it takes to complete the training.

This described procedure is implemented in a script called `train_network.py`. When given a file with scored alignments and files with documents for both the languages, the script trains a network and stores its configuration to a disk on a requested location in the form of an XML file. This allows the method to load and use the trained network at any time. It also enables the system to distribute the trained model to other nodes.

## 3.4 Running

With the second part of the training done, the method has prepared a binary classifier able to decide whether to accept a pair of documents as parallel or not (see Section 3.3.5).

The process of running the trained method on the input data is illustrated in Figure 3.4. The resemblance between this process and the procedure of the second part of the training has been already discussed. Due to the large extent of similarity the shared parts of the process are described briefly as they have been discussed in detail in Section 3.3.

### 3.4.1 Preparing Documents

As mentioned earlier in Section 3.2.1, the input documents have to be preprocessed in the same way as the training parallel corpus. Then, the preprocessed documents have to be split into bins. As already noted (see Section 3.1), when aligning paragraphs from the web, a bin can contain all the paragraphs for both the languages, scraped from one bilingual web domain. In this scenario, the names of the domains can be used as bin identifiers. This restricts the method to align only the paragraphs originating from the same domain. On the other hand, if aligning inseparable data, e.g. documents without a natural distribution into groups, all the documents can be placed into a single bin with an arbitrary identifier. However, the method has better recall when aligning smaller bins. This is caused mainly by Annoy, which has better accuracy when searching through an index with less items.

Our implementation accepts the input dataset in a form of two files, one for each language. It is the same format as for the supervised dataset, described in Section 3.3.1. Listing 3.10 and Listing 3.11 show samples of such a pair of files containing Czech and English paragraphs acquired from the web.

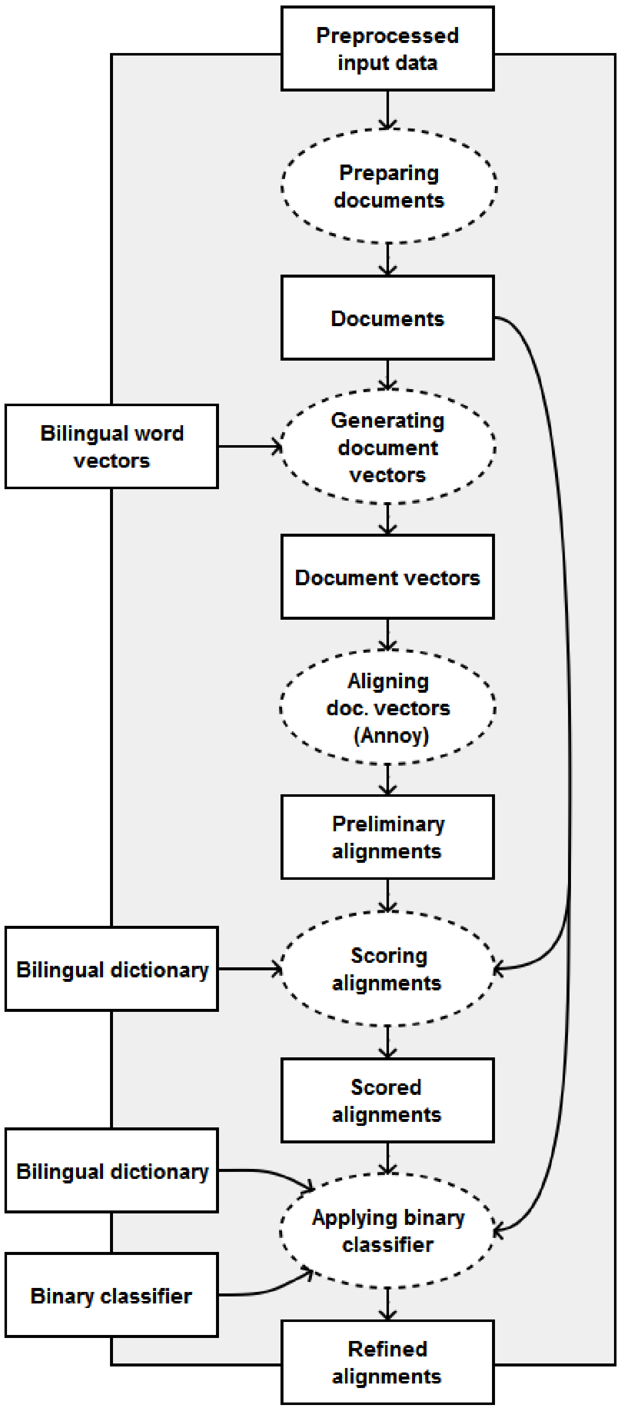
Listing 3.10: Sample from a file with Czech documents (running)

```
europa.eu   154289      v praze se fórum zaměřilo konkrétně na jadernou
     bezpečnost , politiky nukleárního odpadu , možné iniciativy v
    oblasti odborné přípravy a vzdělávání a transparentnosti .
```

### 3.4.2 Applying Binary Classifier

With the input dataset prepared, the process follows with the exact same steps applied to the supervised dataset in the second part of the training. First, vectors

Figure 3.4: Proposed method: running



are generated for all the documents (see Section 3.3.2). Then, the document vectors are aligned by searching for nearest neighbours of all documents in the source language, resulting in preliminary alignments (see Section 3.3.3) and these are subsequently scored (see Section 3.3.4).

Listing 3.11: Sample from a file with English documents (running)

```
europa.eu   1085752     at the prague meeting the forum has been dedicated more particularly to nuclear safety , nuclear waste policies , possible initiatives on training and education as well as in the area of transparency .
```

As a final step, the trained classifier (see Section 3.3.5) is used to obtain the refined alignments. For every document $d$ in the source language and its top candidate $c$ in the target language the trained network is activated as follows:

$$\begin{pmatrix} \text{length\_sim}(d,c) \\ \text{length\_conf}(d,c) \\ \text{weight\_sim}_2(d,c) \\ \text{weight\_conf}_2(d,c) \end{pmatrix} \overset{?}{\rightarrow} \begin{pmatrix} a \\ b \end{pmatrix}$$

The input vector contains the same set of features as in the training. When activated, the second value from the output vector $b \in [0,1]$ represents the confidence of a prediction that the two documents $\langle d, c\rangle$ are parallel. If the confidence $b$ is greater than a user-defined threshold, the document pair $\langle d, c\rangle$ ends up in the resulting refined alignments.

A script called `apply_network.py` implements the described procedure. When given a file containing the network configuration of a classifier, a file with scored alignments and files with documents for both the languages, it creates an output file containing refined alignments. Listing 3.12 shows a sample of its output. The output format follows the convention of the files with preliminary and scored alignments. This time, the last column represents the confidence returned by the classifier. The presented sample shows that the two paragraphs from Listing 3.10 and Listing 3.11 are successfully aligned.

Listing 3.12: Sample from a file with refined alignments (running)

```
europa.eu   154289  1085752     0.9996949388
europa.eu   154287  1085750     0.999479551945
europa.eu   154284  1163960     0.996325842675
europa.eu   154285  1163962     0.996881020525
```

## 3.5 Discussion

The second part of the training process and its resemblance to the procedure of running the trained method may seem cumbersome. However, in case of presence of a different binary classifier able to tell whether the two documents are parallel or not, the whole second part of the training could be excluded and the provided classifier would be utilized.

An alternative classifier could be based on a pre-trained SMT system that would be used to translate the document in the source language into the target one to be compared with the other document. This idea was adapted in the related project focused on mining Wikipedia (see Section 1.4).

Our classifier is more similar to the one used by STRAND (see Section 1.3). The STRAND's classifier is based on a decision tree built over a set of features comparing the similarity between a pair of potentially parallel HTML structures. In contrast to this, our method uses a neural network trained with a set of features designed to compare two potentially parallel plain text documents.

In our method, the second part of the training process simulates the aligning of a supervised dataset with the knowledge of the optimal solution. In the end of the procedure, the model of the classifer is trained. This is a generic approach that can be easily modified. One can redesign the set of features for the classification or completely change the implementation of the learning model.

Inspired by STRAND's classifier, we tried to swap the neural network model with a model based on a decision tree using the same set of features. For these purposes, we used the implementation provided by Scikit-learn [74], a Python machine learning library. The tentative results were poorer; however, we did not spend that much time with trying different parameters and testing.

It is important to note, that the method, in its present form, does not take into account the order of the words when aligning parallel documents. The tf-idf weighting scheme, the scoring function based on the IBM Model 1 and even all the features used by the classifier completely ignore this aspect. Despite the fact that the results are promising, there are places for improvement.

It is also important to highlight the fact that our method is asymmetric regarding the fact that it generates different results if the source and the target languages are swapped. The main reason behind this is that the document vectors associated with the target language are indexed by Annoy while the other vectors for the source language are used as query points for the approximate-nearest-neighbours search. From our experience, a rule of thumb is to choose the language with more documents to be the target one.

The method is designed to be able to run in parallel or distributed environment. Bins with input documents represent independent isolable tasks. These can be distributed across multiple nodes together with the once trained resources needed for the execution. These include bilingual dictionary, bilingual word vectors and the trained classifier.

The key idea behind our method is to use the combination of bilingual word embedding and locality-sensitive hashing. Usually the most expensive operation when solving the task of document alignment is the content comparison of the two documents. A naive approach compares each of the $N$ documents in the source language, with all the $M$ documents in the target language resulting in $N \times M$ comparisons. Our method reduces this number to $N \times k + N$. The $N \times k$ is the number of comparisons needed for the scoring of the $k$ candidates from all the possible $M$ ones as returned by the approximate-nearest-neighbours search and $N$ is the number of comparisons necessary for the final classification of the top candidates.

# Chapter 4

# Prealigned Data (CzEng) Experiment

This chapter describes the first experiment conducted with our method. The experiment involves the prealigned parallel data. This type of data enable us to automatically compare the method's results with the original alignment. Therefore, we can estimate the effectiveness of the proposed method.

For the experiment, we have selected Czech–English language pair. The most important reasons behind this selection are that we can use the available Czech–English parallel data and we understand both these languages well. The parallel corpus used for this experiment consists of all the training sections (packs 00–97) of CzEng 1.0 (see Section 2.2) in the plain text, untokenized format. It contains $14,833,358$ pairs of parallel sentences.

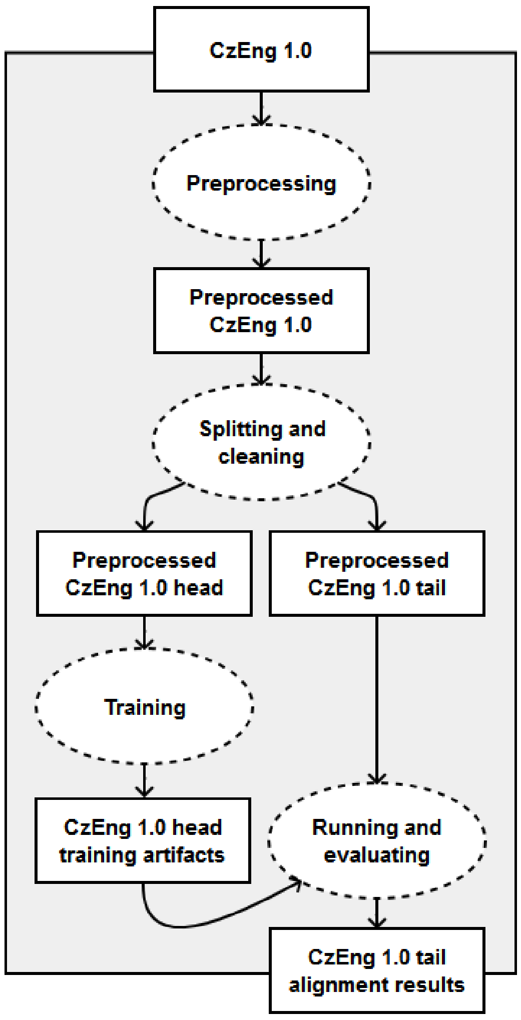
## 4.1 Experiment Procedure

The main idea behind the experiment is the following. CzEng 1.0 is split horizontally in half creating two smaller equally sized parallel corpora. We call these *head* and *tail*. The head is used for the training, while the tail is used for evaluation. The effectiveness of the method is measured after the individual steps of the aligning process. The whole procedure is illustrated in Figure 4.1 and described in the following text.

### 4.1.1 Preprocessing CzEng 1.0

The entire CzEng 1.0 is preprocessed, as a whole. All the sequences of characters that are neither letters from any alphabet nor whitespaces nor even ASCII symbols are replaced with the Unicode symbol • (U+2022). Then, the corpus

Figure 4.1: CzEng experiment


CzEng 1.0
Preprocessing
Preprocessed CzEng 1.0
Splitting and cleaning
Preprocessed CzEng 1.0 head
Preprocessed CzEng 1.0 tail
Training
CzEng 1.0 head training artifacts
Running and evaluating
CzEng 1.0 tail alignment results

is tokenized and subsequently lowercased. The tokenization is done using MorphoDiTa (see Section 2.3). When running MorphoDiTa's script `run_tokenizer` on the Czech and English parts of the parallel corpus, the argument `--tokenizer` is set to `czech` and `english` respectively.

### 4.1.2 Splitting and Cleaning CzEng 1.0

Preprocessed CzEng 1.0 is split horizontally exactly in half creating two separate parallel corpora, i.e. head and tail. The head is cleaned by excluding all pairs where either of the two sentences contains more than 50 tokens (286, 318 pairs are filtered out) or does not contain a single letter from any alphabet (more 103, 251 are discared). The tail is cleaned by only the latter of the two mentioned filters (104, 103 pairs are exluded). The pairs containing overly long sentences are removed from the head to get better quality word alignment, which heavily affects the whole training process.

## 4.1.3 Training Part I: Dictionary, Word Vectors

As the first step of the training, SyMGIZA++ is applied (see Section 3.2.2) to calculate the word alignment for the head. This includes the standard workflow with the scripts: `plain2snt`, `snt2cooc` and `mkcls`, where we request 2 optimization runs (`-n2`). For the execution of `symgiza`, we use the standard settings listed in Table 4.1. From our experience, the method of final symmetrization yielding the best results is the "union" method (`-alig union`).

Table 4.1: CzEng experiment: SyMGIZA++ settings

| **Description** | **Argument** |
|---|---|
| Number of threads | `-ncpus 4` |
| Number of Model 1 iterations | `-m1 5` |
| Number of Model 2 iterations | `-m2 5` |
| Number of Model 3 iterations | `-m3 5` |
| Number of Model 4 iterations | `-m4 5` |
| Number of HMM iterations | `-mh 5` |
| Dump Model 1 after 5th iteration | `-t1` |
| Dump Model 2 after 5th iteration | `-t2` |
| Dump Model 3, 4, 5 after 5th iteration | `-t345` |
| Dump Model HMM after 5th iteration | `-th` |
| Symmetrize Model 1 after 5th iteration | `-m1symfrequency 5` |
| Symmetrize Model 2 after 5th iteration | `-m2symfrequency 5` |
| Symmetrize Model 3, 4, 5 after 5th iteration | `-m345symfrequency 5` |
| Symmetrize Model HMM after 5th iteration | `-mhsymfrequency 5` |
| Symmetrization "t" tables multiplier | `-tm 2` |
| Run final symmetrization | `-es 1` |
| Use Union method in final symmetrization | `-alig union` |
| Omit Diagonal option in final symmetrization | `-diagonal no` |
| Omit Final option in final symmetrization | `-final no` |
| Omit Both option in final symmetrization | `-both no` |
| Probability for empty words | `-emprobforempty 0.0` |
| Probability smoothing value | `-probsmooth 1e-7` |

Subsequently, the bilingual dictionary is generated (see Section 3.2.3) using the script `merge_param.py`. The created dictionary contains $11,567,603$ entries.

Then, bivec is applied (see Section 3.2.4) to create the bilingual word vectors. Prior to the execution, the training dataset is preprocessed by replacing all the Unicode symbols • (U+2022) symbols with `<unk>` and the sequences consisting of numbers with zero symbol. We use the standard settings listed in Table 4.2. The resulting output contains $367,393$ vectors for the Czech and $179,869$ for the English part of the corpus.

Table 4.2: CzEng experiment: bivec settings

| Description | Argument |
|---|---|
| Language code of source language | `-src-lang en` |
| Language code of target language | `-tgt-lang cs` |
| Use the provided word alignment | `-align-opt 1` |
| Cross-lingual learning rate multiplier | `-bi-weight 1.0` |
| Use biskip model | `-cbow 0` |
| Discard less appearing words | `-min-count 3` |
| Size of the output vectors | `-size 40` |
| Maximal skip length between words | `-window 5` |
| Number of negative examples | `-negative 5` |
| Save output in textual format | `-binary 0` |
| Do not use hierarchical softmax | `-hs 0` |
| Threshold for high-frequency source words | `-sample 1e-4` |
| Threshold for high-frequency target words | `-tgt-sample 1e-4` |
| Number of threads | `-threads 4` |
| Do not evaluate results | `-eval 0` |
| Number of iterations | `-iter 10` |

### 4.1.4 Training Part II: Classifier

The pairs of parallel sentences from the head are distributed into artificial bins to form a supervised dataset for the training of the classifier (see Section 3.3.1). Each bin contains $50,000$ pairs of the parallel documents, i.e. $100,000$ individual documents. We consider this value to be an upper-bound estimate of an average number of paragraphs in either of the two languages located on an ordinary Czech–English web domain. The created dataset consists of 141 bins. The last bin is an exception containing only $27,110$ pairs.

The document vectors are generated (see Section 3.3.2) using the script `create_docvec.py`. The supervised dataset contains $7,027,110$ pairs of the parallel documents for which the script generates $6,467,817$ vectors for the Czech and $6,420,329$ vectors for the English documents. The numbers differ because the script discards duplicated documents within the bins.

The preliminary alignments are created (see Section 3.3.3) running the script `align_docvec.py`. For every Czech document a list of 20 English candidate documents is created. Annoy (see Section 2.6) is set to build search indices with 500 trees and when performing a search it is requested to inspect $500 \times 20 \times 2 = 20,000$ nodes. These settings greatly affect the results. We follow the recommendations [55] to set the number of trees as large as possible given the amount of available memory and the number of nodes to be inspected as large as possible given the amount of available computational time.

The alignments are scored (see Section 3.3.4) with the script `score_align.py`. In this experiment, we use the expected ratio of the documents's lengths $\mu_{cs\rightarrow en} =$

1.08 with the standard deviation $\sigma_{cs \to en} = 0.28$. These values were estimated using the pairs of parallel sentences in the head.

The classifier is trained (see Section 3.3.5) with the script `train_network.py`. The script creates the training dataset by randomly selecting approximately 20% of all the available pairs of a Czech document with its top English candidate. Additionally, the selection contains nearly as many parallel pairs as non-parallel. The network model is provided by PyBrain (see Section 2.7). For completeness, let us describe its configuration; however, we shall not go into details [58]. The network is a multilayer perceptron learning through the backwards error propagation. It has 4 input, 16 hidden and 2 output neurons. The input neurons are linear, the hidden layer uses sigmoid function and the output layer uses softmax function. The network is trained for 20 epochs with the 1% learning rate.

### 4.1.5 Running

The trained system is used to search for sentence pairs in the tail. The pairs of parallel sentences from the tail are distributed into artificial bins in a same manner as those from the head. Also, in this case, each bin contains 50,000 pairs of the parallel documents, i.e. 100,000 individual documents. In this scenario, each bin simulates a web domain with 50,000 Czech and 50,000 English paragraphs that we want to align. In contrast to real websites, these are perfectly parallel, all pages are available in both languages. This way, 147 bins are created. The last bin is an exception containing only 12,576 pairs. For the purposes of the experiment, the original alignment of the tail is forgotten to not affect anyhow the evaluation process.

For the input dataset, the procedure follows with the exact same steps as for the supervised dataset in the second part of the training. Vectors are generated for all the documents. Using Annoy, the document vectors are aligned into preliminary alignments and these are subsequently scored. All of the settings remain unchanged. The input dataset consists of 7,312,576 pairs for which 6,750,340 vectors for the Czech and 6,703,831 vectors for the English documents are generated. The differences between these numbers are caused again by the presence of duplicate documents within the bins.

In the last step, the trained binary classifier is applied (see Section 3.4.2) to obtain the refined alignments for the input dataset. This is done using the script `apply_network.py`. The confidence threshold of the classifier is set to 50%. The refined alignments represent a subset of all the pairs of Czech documents with their top English candidates that the classifier accepts to be parallel.

## 4.2 Experiment Results

With the entire procedure of the experiment described, let us examine the results. The effectiveness of the method is measured after the individual steps of the aligning process. We present the results for the tail from the evaluation but also for the head from the second part of the training. This way we can compare the difference between the results when aligning the data used also in the first part of the training and the data not covered in the training.

Table 4.3 shows the quality of the preliminary alignments. The row with an index value $k$ shows how many times the parallel document ends up as a $k^{\text{th}}$ best candidate. The one with an index value $k \leq 20$ tells how many times the parallel document appears somewhere in the candidate list and the row with $k > 20$ shows how many times the parallel document is not in a candidate list at all.

Table 4.3: CzEng experiment: preliminary alignments

| | **Head (Training)** | | **Tail (Evaluation)** | |
|---:|---:|---:|---:|---:|
| **Index** | **Count** | **Ratio (%)** | **Count** | **Ratio (%)** |
| 1 | 3,310,898 | 51.19 | 3,395,454 | 50.30 |
| 2 | 477,165 | 7.38 | 499,868 | 7.41 |
| 3 | 226,139 | 3.50 | 237,930 | 3.52 |
| 4 | 144,859 | 2.24 | 152,567 | 2.26 |
| 5 | 105,706 | 1.63 | 111,802 | 1.66 |
| 6 | 83,212 | 1.29 | 87,839 | 1.30 |
| 7 | 68,488 | 1.06 | 72,062 | 1.07 |
| 8 | 57,827 | 0.89 | 60,867 | 0.90 |
| 9 | 49,544 | 0.77 | 53,050 | 0.79 |
| 10 | 44,125 | 0.68 | 46,556 | 0.69 |
| 11 | 39,279 | 0.61 | 41,700 | 0.62 |
| 12 | 35,638 | 0.55 | 37,677 | 0.56 |
| 13 | 32,453 | 0.50 | 34,069 | 0.50 |
| 14 | 29,829 | 0.46 | 31,497 | 0.47 |
| 15 | 27,548 | 0.43 | 28,954 | 0.43 |
| 16 | 25,280 | 0.39 | 26,995 | 0.40 |
| 17 | 23,588 | 0.36 | 24,964 | 0.37 |
| 18 | 21,875 | 0.34 | 23,201 | 0.34 |
| 19 | 20,635 | 0.32 | 22,024 | 0.33 |
| 20 | 19,639 | 0.30 | 20,736 | 0.31 |
| $\leq 20$ | 4,843,727 | 74.89 | 5,009,812 | 74.22 |
| $> 20$ | 1,624,090 | 25.11 | 1,740,528 | 25.78 |
| **Total** | 6,467,817 | 100.00 | 6,750,340 | 100.00 |

The results for the head show, that 74.89% Czech documents have their parallel English document included in the candidate list. This number is similar also for the tail, where it equals 74.22%. The difference is surprisingly small, as the training does not know anything about the tail. By further inspecting the results for the tail, we can observe, that of all the situations when the parallel document appears somewhere in the candidate list, in 67.78% it is the top one and in 94.18% it is included in the top 10. This means that if we reduce the size of the search from 20 to 10, we can still expect approximately 69.89% Czech documents to have the parallel English document somewhere in the candidate list. Reducing the size of the query increases the speed.

Table 4.3 shows the quality of the scored alignments. The process of scoring does not change the number of the Czech documents having a parallel English document present in the candidate list. Actually, it only reorders the candidate lists. The intention is to push the parallel documents within their candidate lists to the top as much as possible. The results show that the scoring function based on IBM Model 1 combined with the length comparison is effective. Again, the situation is obviously slightly better for the head. This is mainly caused by the fact, that the bilingual dictionary used in process is built-up from the head only. The results for the tail show that of all the times that parallel document appears in the candidate list, in 96.07% it is the top one. We consider this a good reason why should the following process consider only the top candidates.

Table 4.4: CzEng experiment: scored alignments

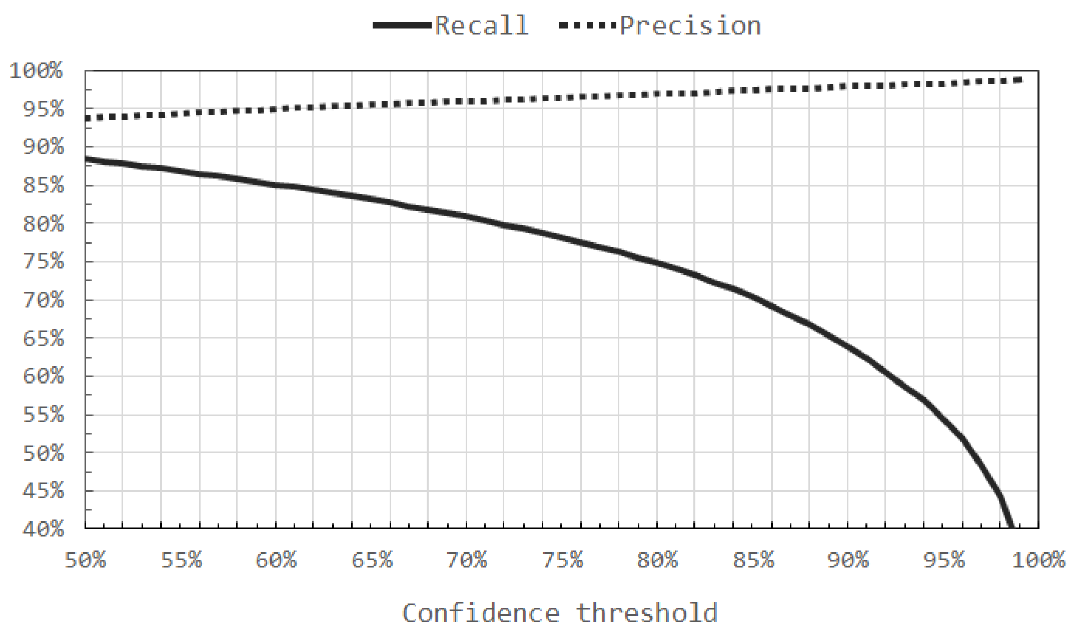
| | **Head (Training)** | | **Tail (Evaluation)** | |
|---|---|---|---|---|
| **Index** | **Count** | **Ratio (%)** | **Count** | **Ratio (%)** |
| 1 | 4,689,885 | 72.51 | 4,812,681 | 71.30 |
| 2 | 90,631 | 1.40 | 112,411 | 1.67 |
| 3 | 24,786 | 0.38 | 32,603 | 0.48 |
| 4 | 12,033 | 0.19 | 16,309 | 0.24 |
| 5 | 7,198 | 0.11 | 9,774 | 0.14 |
| 6 | 4,728 | 0.07 | 6,577 | 0.10 |
| 7 | 3,345 | 0.05 | 4,608 | 0.07 |
| 8 | 2,505 | 0.04 | 3,475 | 0.05 |
| 9 | 1,955 | 0.03 | 2,639 | 0.04 |
| 10 | 1,544 | 0.02 | 1,980 | 0.03 |
| 11 | 1,190 | 0.02 | 1,516 | 0.02 |
| 12 | 971 | 0.02 | 1,270 | 0.02 |
| 13 | 746 | 0.01 | 957 | 0.01 |
| 14 | 552 | 0.01 | 781 | 0.01 |
| 15 | 430 | 0.01 | 619 | 0.01 |
| 16 | 407 | 0.01 | 520 | 0.01 |
| 17 | 280 | 0.00 | 407 | 0.01 |
| 18 | 244 | 0.00 | 313 | 0.00 |
| 19 | 179 | 0.00 | 232 | 0.00 |
| 20 | 118 | 0.00 | 140 | 0.00 |
| $\leq 20$ | 4,843,727 | 74.89 | 5,009,812 | 74.22 |
| $> 20$ | 1,624,090 | 25.11 | 1,740,528 | 25.78 |
| **Total** | 6,467,817 | 100.00 | 6,750,340 | 100.00 |

The refined alignments are obtained with the 50% confidence threshold. By gradually increasing the threshold and further filtering the alignments we measure the recall and the precision of the classifier at different confidence levels. The results are summarized in Table 4.5. The first column represents the confidence threshold. The second column represents the number of the document pairs correctly identified to be parallel, i.e. the number of true positives. The next column shows the number of false positives, which is the number of document pairs identified as parallel, but in fact they are not. The recall listed in the table is relative to the input of the classification process. The precision is the ratio of the true positives to all the positives. Figure 4.2 shows how the recall and the precision change with respect to the confidence threshold of the classifier.

Table 4.5: CzEng experiment: classifier effectiveness

| Conf.(%) | True Pos. | False Pos. | Recall (%) | Precision (%) |
|---:|---:|---:|---:|---:|
| 50.00 | 4,254,069 | 284,174 | 88.39 | 93.74 |
| 55.00 | 4,179,076 | 248,921 | 86.83 | 94.38 |
| 60.00 | 4,095,812 | 217,081 | 85.10 | 94.97 |
| 65.00 | 4,000,964 | 188,595 | 83.13 | 95.50 |
| 70.00 | 3,892,099 | 162,537 | 80.87 | 95.99 |
| 75.00 | 3,761,728 | 138,619 | 78.16 | 96.45 |
| 80.00 | 3,599,299 | 114,985 | 74.79 | 96.90 |
| 85.00 | 3,385,007 | 89,813 | 70.34 | 97.42 |
| 90.00 | 3,073,706 | 64,730 | 63.87 | 97.94 |
| 95.00 | 2,621,246 | 46,522 | 54.47 | 98.26 |
| 99.00 | 1,808,067 | 23,028 | 37.57 | 98.74 |

Figure 4.2: CzEng experiment: classifier effectiveness



The overall effectiveness of our method is listed in Table 4.6. These values are measured using the refined alignments without any form of additional filtering.

Table 4.6: CzEng experiment: overall effectiveness

| | |
|---|---|
| **Recall (%)** | 63.02 |
| **Precision (%)** | 93.74 |

## 4.3 Experiment Time Duration

The computer used for the experiment execution has Intel® Xeon® CPU E5-2630 v3 (20 MB Cache, 2.40 GHz) and 128 gigabytes (GB) of memory. Table 4.7 lists the approximate time durations of the individual steps of the experiment.

Table 4.7: CzEng experiment: time duration

| **Activity** | **Duration (hh:mm)** |
|---|---|
| **Preprocessing** | |
| Tokenizing and lowercasing | 00:08 |
| Splitting and cleaning | 00:05 |
| **Training part I** | |
| Applying SyMGIZA++ | 13:21 |
| Generating dictionary | 00:10 |
| Applying bivec | 01:01 |
| **Training part II** | |
| Generating document vectors | 00:37 |
| Aligning document vectors (Annoy) | 05:52 |
| Scoring alignments | 02:49 |
| Training network classifier | 01:29 |
| **Evaluation** | |
| Generating document vectors | 00:45 |
| Aligning document vectors (Annoy) | 07:04 |
| Scoring alignments | 04:10 |
| Applying network classifier | 00:47 |

## 4.4 Extension: Lemmatization

As already noted (see Section 3.2.1), we believe that preprocessing of both training and input data plays an important role in our method. The extension of the experiment described is this section is conducted with a goal to determine whether the lemmatization helps the method to achieve better quality results for the Czech–English language pair.

The procedure of the extended experiment is almost completely the same as in the original experiment. The only difference is in the preprocessing of the data. The lemmatization is added between the tokenization and lowercasing. The lemmatization is done with MorphoDiTa (see Section 2.3). When running the script `run_tagger` on the Czech and English parts of the parallel corpus, it is provided with the models `czech-morfflex-pdt-131112.tagger-best_accuracy` [42] and `english-morphium-wsj-140407.tagger` [43] respectively.

Table 4.8: CzEng experiment (extended): scored alignments

| | **Head (Training)** | | **Tail (Evaluation)** | |
|---|---|---|---|---|
| **Index** | **Count** | **Ratio (%)** | **Count** | **Ratio (%)** |
| 1 | 4,639,484 | 72.10 | 4,813,195 | 71.64 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $\leq 20$ | 4,813,165 | 74.80 | 5,022,948 | 74.76 |
| $> 20$ | 1,621,610 | 25.20 | 1,695,537 | 25.24 |
| **Total** | 6,434,775 | 100.00 | 6,718,485 | 100.00 |

With the lemmatization included, the dictionary built-up from the head contains only $6,225,379$ entries (53.82% of the original size). The extension also reduces the number of word vectors produced by bivec, which is $173,861$ (47.32%) vectors for the Czech and $163,993$ (91.17%) for the English part of the corpus.

Table 4.8 shows how the lemmatization affects the results. It shows the quality of the scored alignments. Although the table is not complete, it contains all the relevant data. The results show minimal improvement. For the tail, the amount of Czech documents having the parallel English document in the candidate list is increased by only 0.54%. Additionally, Table 4.9 lists the overall effectiveness of the method with lemmatization included. Again, the results are very similar.

Table 4.9: CzEng experiment (extended): overall effectiveness

| | |
|---|---|
| **Recall (%)** | 63.28 |
| **Precision (%)** | 94.32 |

Lemmatization can be thought of as a many-to-one mapping for words. By applying a transformation based on such a mapping we reduce the number of words within a corpus. With less words in the corpus, it is easier for the method to learn the associations between the words; however, the individual sentences are losing their contextual diversity. Apparently, the CzEng 1.0 data are already sufficiently large and the union word alignments do not benefit significantly from the denser statistics due to lemmatization.

# Chapter 5

# Web Data (Common Crawl) Experiment

In this chapter, the second experiment conducted with our method is discussed. Unlike the first experiment (see Chapter 4) this one deals with the non-parallel, real-word, noisy data acquired from the web. It illustrates effectiveness of our method in a typical situation for which it is designed.

The selected language pair is the same as in the first experiment, i.e. Czech–English. This experiment's procedure utilizes the training artifacts created in the first experiment, namely the dictionary, bilingual word vectors and the trained classifier. The input data are obtained from the July 2015 dataset provided by the Common Crawl organization (see Section 2.8).

The following text describes the procedure of the experiment with an example approach to the task of mining parallel corpus from the Common Crawl dataset for a specific language pair. It also discusses the manually evaluated quality of the acquired Czech–English parallel corpus.

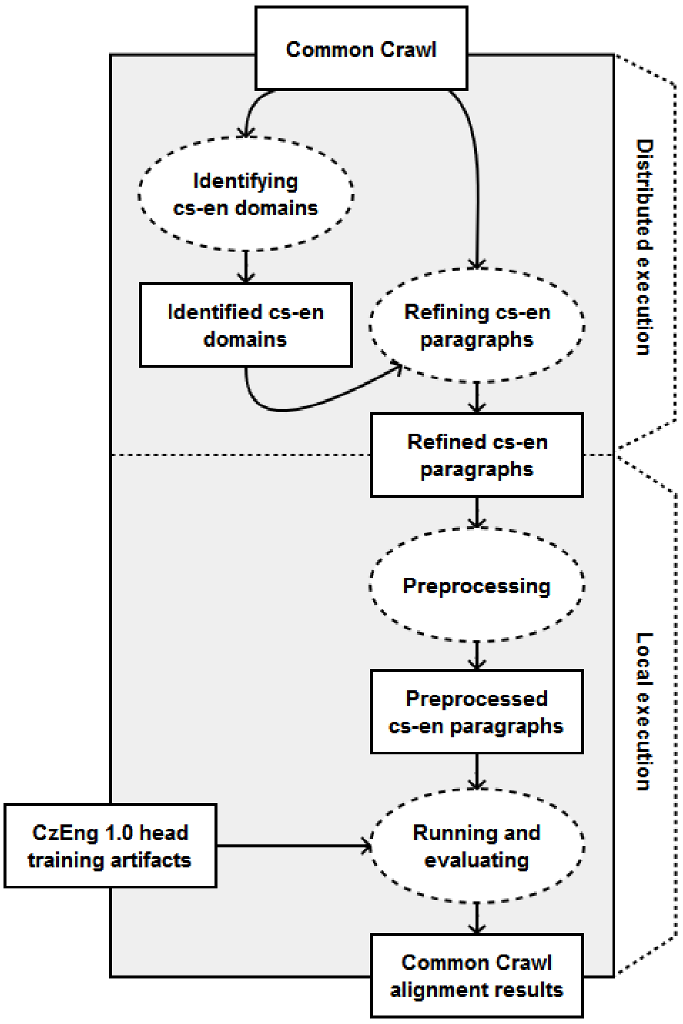
## 5.1 Experiment Procedure

The July 2015 dataset consists of approximately 1.84 billions of crawled web pages and it takes about 149 terabytes (TB) of disk space in the uncompressed WARC format. To process this large volume of data we use Hadoop (see Section 2.9) cluster provided by MetaCentrum (see Section 2.11). The dataset is available in a form of $33,957$ WARC files compressed by GNU zip (gzip)[1]. Each of these files has less than 950 megabytes (MB) and is accessible via its own URL. The list of these URLs can be obtained at the official website of the Common Crawl [11] organization. The total size of the July 2015 dataset in the compressed format is approximately 28.5 TB.

[1]http://www.gzip.org/ (accessed April 23, 2016)

One of the options to get all the files into HDFS (see Subsection 2.9.1), is to download them in sequence with cURL[2] using one node of the cluster. During the process, every downloaded file can be immediately copied into HDFS using the command `hadoop fs -copyFromLocal` and afterwards deleted locally. To reduce the time, the process can use multiple threads. In our environment, we use 5 threads and the described process takes about 17 days to complete. Such a long process can be interrupted by many circumstances (e.g. cluster maintenance). Therefore, our script first checks which of the files are already present in HDFS using the command `hadoop fs -ls` to avoid repetitive downloading. Hadoop allows the user to process the WARC files compressed by gzip. We use this feature to reduce the required disk space at the cost of slower execution.

Figure 5.1: Common Crawl experiment



Common Crawl
Identifying cs-en domains
Identified cs-en domains
Refining cs-en paragraphs
Refined cs-en paragraphs
Preprocessing
Preprocessed cs-en paragraphs
CzEng 1.0 head training artifacts
Running and evaluating
Common Crawl alignment results
Distributed execution
Local execution

With the entire July 2015 dataset stored in HDFS, the experiment illustrated in Figure 5.1 is applied. The procedure starts with the distributed execution running two MapReduce (see Subsection 2.9.2) jobs. This creates a dataset containing Czech and English paragraphs from the web domains identified to be bilingual.

[2]`https://curl.haxx.se/` (accessed April 23, 2016)

The paragraphs are then aligned with our method in a local execution and the results are evaluated. The whole process is described as follows.

### 5.1.1 Distributed Execution

The part of the experiment's procedure executed in the distributed environment consists of two MapReduce jobs. The first job identifies the web domains containing paragraphs in both the languages we are interested in. In order to enable the MapReduce framework to read and write the WARC files properly we use WARC-Hadoop (See Section 2.10).

Let us describe the implementations of the mapper and the reducer in the first MapReduce job. The mapper is called *WarcTextMapper*. It iterates through the WARC records processing only those which represent HTTP responses with `text/html` content type, i.e. web pages. For every web page it resolves the character encoding and transforms its HTML structure into XHTML using jsoup (see Section 2.12). Then, the textual contents of all the `<p>` HTML tags representing paragraphs are parsed and all those having less than 100 characters are discarded. The shorter paragraphs are discarded because it is difficult to detect their language reliably. For each paragraph the language is identified with a certain confidence using language-detector (see Section 2.13) . The mapper outputs a key-value pair for each paragraph where the language is detected as one of the two languages we are interested in with the confidence at least 99%. The structure of an output key-value pair is following:

- Key: web domain name.
- Value:
  - language detected by language-detector;
  - confidence returned by language-detector;
  - URL associated with the paragraph;
  - textual content of the paragraph.

The implementation of the reducer is called *WarcDomainReducer*. For a given web domain (i.e. key) it receives the values representing all the paragraphs for the domain emitted by *WarcTextMapper*. The reducer iterates over all these values counting the number of the unique paragraphs (using hashing) and their total length for both the languages separately. It also counts the number of the unique URLs. For every domain having at least one Czech and one English paragraph the reducer outputs the key-value pair as follows:

- Key: web domain name.
- Value:
    - number of unique URLs;
    - number of unique English paragraphs;
    - total length of unique English paragraphs;
    - number of unique Czech paragraphs;
    - total length of unique Czech paragraphs.

The first MapReduce job creates a list of web domains having at least some Czech and English content. Listing 5.1 shows a sample from its output file. The format contains tab-separated values, where the first column is a web domain, which is the key. The other columns represent the individual items of the value in the order they were listed. The list contains 12, 144 domains.

Listing 5.1: sample from a file with identified domains

```
www.meuslany.cz     2   6    2813     16   4509
www.mff.cuni.cz     7   37   14946    39   18162
```

As the next step, a filter is applied on the output of the first MapReduce job reducing the number of web domains to be considered for the further processing. The filter requires from a domain to meet the following condition:

$$\frac{\min(N_{cs}, N_{en})}{\max(N_{cs}, N_{en})} > 1\%,$$

where $N_{cs}$ and $N_{en}$ are the numbers of Czech and English paragraphs for the domain respectively. The filtering discards all the domains having very unbalanced language distribution. The output contains 8, 750 identified bilingual domains.

The second MapReduce job extracts the Czech and English paragraphs for all the identified bilingual web domains. It uses the same mapper as the first job; however, the implementation of the reducer is different. We call it *WarcTextReducer*. In order to provide the reducer with the file containing the domains Hadoop Distributed Cache is utilized. It is a facility provided by the MapReduce framework enabling the user to cache files for a MapReduce job. Once a file is cached for a job the framework makes it available locally at each node of the cluster for the time of the execution allowing mappers and reducers to read the cached file. When initializing, *WarcTextReducer* reads the entire file with the bilingual domains creating a set of hash codes for all their names. When running, it outputs only those incoming key-value pairs emitted by *WarcTextMapper* that are associated with one of the domains. Additionally, all the values representing duplicate paragraphs are discarded. The structure of the output key-value pairs emitted by both *WarcTextMapper* and *WarcTextReducer* is identical.

Listing 5.2 and Listing 5.3 show samples of two lines from the output file of the second MapReduce job. These two lines contain parallel paragraphs that are mutual translations. The single output file contains the paragraphs for both the languages. The format is a list of tab-separated values. The first column is a web domain, which is the key. The following columns represent the individual items of the value in the order they were listed in the text describing the structure of the output key-value pairs emitted by *WarcTextMapper*. The output file contains $5,931,091$ paragraphs for both the languages, namely $801,116$ Czech and $5,129,975$ English. The Czech and English paragraphs originate from $127,570$ and $744,074$ unique URLs, respectively. The average length of a Czech paragraph is 352.78 characters, while for an English one, it is 417.03.

Listing 5.2: sample from a file with extracted paragraphs (Czech)

```
czechfolks.com  cs  0.99999445885996    http://czechfolks.com
    /2009/11/25/how-well-do-you-know-the-czech-republic-sweepstakes-
    results-jak-dobre-znate-ceskou-republiku-a-vysledky-souteze/   Zde
    je otázka, kterou jsme položili a její správná odpověd’: Otázka: Kdo
     byl Karel IV? Odpověd’: Český král a římský císař.
```

Listing 5.3: sample from a file with extracted paragraphs (English)

```
czechfolks.com  en  0.9999980969126909  http://czechfolks.com
    /2009/11/25/how-well-do-you-know-the-czech-republic-sweepstakes-
    results-jak-dobre-znate-ceskou-republiku-a-vysledky-souteze/   Here
    is the question that we asked with the correct answer: Question: Who
     was Charles IV? Answer: The king of Bohemia and Holy Roman Emperor.
```

### 5.1.2 Local Execution

The file containing the extracted paragraphs is transferred from HDFS to a one node of the cluster. The process continues in a single-node execution. The input dataset for our method is formed by distributing the paragraphs into the bins according to domain names. For each bilingual web domain, a bin is created with all the associated paragraphs in both the languages. This restricts the method to align only the paragraphs belonging to the same domain.

The rest of the procedure is similar as for the tail in the first experiment with CzEng 1.0 (see Section 4.1.5). The input dataset is preprocessed by tokenization and lowercasing and our method is executed creating refined alignments for the paragraphs. The method is provided with the training artifacts created during the first experiment using the head of CzEng 1.0, namely the dictionary, bilingual word vectors and the trained classifier. The only difference in the settings is the changed confidence threshold for the classifier, which is required to be 99%. The precision is favoured over the recall.

It is important to note that for this experiment the Czech language is selected as the source language. Therefore, the paragraph vectors associated with the English language are the ones indexed by Annoy. This selection follows the already mentioned rule of thumb (see Section 3.5) to let the method index the document vectors for the language having more documents.

## 5.2 Experiment Results

Table 5.1 lists the most frequent web domains appearing in the extracted pairs of paragraphs. The full list contains 2,178 domains having altogether 114,711 pairs of aligned paragraphs. This means, that the output of our method contains an alignment for 14,32% of all the extracted Czech paragraphs. The extracted paragraph-aligned parallel corpus contains in total 7,235,908 Czech and 8,369,870 English tokens.

Table 5.1: Common Crawl experiment: web domains of paragraph pairs

| **Source Domain** | **Paragraph Pairs** | **Ratio (%)** |
|---|---:|---:|
| europa.eu | 23457 | 20.45 |
| eur-lex.europa.eu | 15037 | 13.11 |
| windows.microsoft.com | 11905 | 10.38 |
| www.europarl.europa.eu | 8560 | 7.46 |
| www.project-syndicate.org | 2210 | 1.93 |
| www.debian.org | 2191 | 1.91 |
| support.office.com | 1908 | 1.66 |
| www.esa.int | 1308 | 1.14 |
| www.eea.europa.eu | 1299 | 1.13 |
| www.muni.cz | 1206 | 1.05 |
| ⋮ | ⋮ | ⋮ |
| **Total** | 114,711 | 100.00 |

The quality of the extracted corpus is evaluated manually on a set of randomly selected 500 paragraph pairs. The inspected pairs are divided into few categories. The results of this subjective evaluation are displayed in Table 5.2. The pair of paragraphs is considered to be a human translation if it seems like created by a human. These are the most favorable ones. If the translation of the pair seems cumbersome, it is labeled as a product of machine translation. The partial match represents a situation when a paragraph is a translation of only a part of the other having some extra content. Everything else is labeled as a mismatch.

Table 5.2: Common Crawl experiment: evaluation (500 paragraph pairs)

| **Category** | **Count** | **Ratio (%)** |
|---|---:|---:|
| Human translation | 466 | 93.20 |
| Machine translation | 7 | 1.40 |
| Partial match | 13 | 2.60 |
| Mismatch | 14 | 2.80 |
| **Total** | 500 | 100.00 |

To estimate the precision of our method, let us consider the pairs of paragraphs belonging to the categories of human and machine translation as the true positives. This way all the other pairs are regarded as the false positives. Table 5.3 shows how the precision of the method changes with respect to the confidence threshold of the classifier.

Table 5.3: Common Crawl experiment: precision (500 paragraph pairs)

| **Conf.(%)** | **True Pos.** | **False Pos.** | **Precision (%)** |
|---:|---:|---:|---:|
| 99.00 | 473 | 27 | 94.60 |
| 99.10 | 464 | 25 | 94.89 |
| 99.20 | 456 | 23 | 95.20 |
| 99.30 | 446 | 20 | 95.71 |
| 99.40 | 435 | 17 | 96.24 |
| 99.50 | 415 | 16 | 96.29 |
| 99.60 | 404 | 16 | 96.19 |
| 99.70 | 386 | 12 | 96.98 |
| 99.80 | 357 | 7 | 98.08 |
| 99.90 | 286 | 3 | 98.96 |

The task of estimating the recall of our method is more difficult. We would need to manually inspect all the paragraphs from every web domain selected for the evaluation. Therefore, let us perform the recall estimation for only one web domain with smaller number of paragraphs present in the input dataset. The selected web domain is `www.csa.cz`, the official website of Czech Airlines containing human-translated content. For the domain, the input dataset contains 68 Czech and 87 English paragraphs. These are manually aligned, creating what we consider to be the ideal alignments. When evaluated, the ideal alignments contain 44 paragraph pairs, of which 42 appear also in the corpus extracted by our method. Additionally, the corpus include 1 extra pair subjectively regarded as mismatch. Table 5.4 shows the effectiveness of our method evaluated for the `www.csa.cz` web domain. The results are satisfactory; however, `www.csa.cz` is a website with a large proportion of the content provided in both the languages. This fact makes it one of the cleaner sources of parallel Czech–English content.

Table 5.4: Common Crawl experiment: effectiveness (`www.csa.cz`)

| | |
|---|---|
| **Recall (%)** | 95.45 |
| **Precision (%)** | 97.67 |

## 5.3 Experiment Time Duration

The cluster used for the distributed execution of the MapReduce jobs is described in Section 2.11. The rest of the procedure, i.e. the local execution, is done on one node of the cluster having Intel® Xeon® CPU E5-2630 v3 (20 MB Cache, 2.40 GHz) and 128 gigabytes (GB) of memory. Table 5.5 contains the approximate time durations of the individual steps of the experiment.

Table 5.5: Common Crawl experiment: time duration

| **Activity** | **Duration (hh:mm)** |
|---|---|
| **MapReduce framework** | |
| Identifying cs-en domains | 11:58 |
| Refining cs-en paragraphs | 11:38 |
| **Local Execution** | |
| Tokenization and lowercasing | 00:09 |
| Generating document vectors | 00:58 |
| Aligning document vectors (Annoy) | 01:13 |
| Scoring alignments | 03:42 |
| Applying network classifier | 00:39 |

# Conclusions and Future Work

The main objectives of our thesis were to propose a new method for bilingual document alignment, applicable in the field of mining parallel data from the web, and to conduct experiments with the method, presenting its capabilities and effectiveness. The method is partially inspired by the related work, but it is based on a different approach.

The majority of the known methods search for pairs of parallel web pages by the similarity of their HTML structures. They also rely on HTML structures when aligning contents of web pages already identified as parallel. In contrast to these methods, our method does not depend on any kind of page structure comparison at all.

The proposed method is supervised and generic in nature. First, it needs to be trained using a provided sentence-aligned parallel corpus for a given language pair. When trained, the method solves the task of bilingual document alignment—given a set of documents in the two languages, it finds the pairs of parallel ones. With the method applied to the task of mining parallel corpora from the web, we are able to effectively identify the pairs of parallel segments (i.e. paragraphs) located anywhere on the pages of a web domain, regardless of their structure.

The most important step of our method is based on the combination of recent ideas, namely the bilingual extension of word2vec—bivec—and the locality-sensitive hashing (LSH). The method uses bivec to learn vectors for the words in both languages from a provided training parallel corpus. The word vectors are used to calculate the aggregate vectors for the documents to be aligned. These document vectors belong to a common vector space, where pairs of parallel documents tend to have similar vectors. To effectively search the space of document vectors for parallel document candidates, we use Annoy—an implementation of the approximate-nearest-neighbours search based on SimHash, one of the LSH algorithms. In order to decide whether to accept a pair of document and its candidate as parallel or not, a binary classifier is trained using the provided training parallel corpus. The classifier model is based on a neural network, and it uses a set of defined features for the classification.

# Results

To verify the idea of our method, we have performed two experiments focused on the Czech–English language pair. The first one uses prealigned data, and its results are evaluated automatically. It simulates a scenario with 147 web domains, each of them containing approximately 50, 000 Czech and 50, 000 English paragraphs to be aligned. In the experiment, the ideal solution consists of an alignment for each paragraph. The results of the experiment are 63.28% recall at 94.32% precision. In an extension of the experiment, we have observed that including lemmatization into the standard preprocessing (tokenization and lowercasing) of both the training and input data does not improve the quality of the resulting alignments notably.

The second experiment involves the real-world data provided by the Common Crawl Foundation. It demonstrates an application of our method to mining parallel corpora from a hundreds of terabytes (TB) large set of web-crawled data. We have managed to extract the Czech–English parallel corpus from a 149 TB large dataset consisting of 1.84 billions of web pages. By implementing and running two MapReduce jobs, we were able to identify 8, 750 web domains having detectable amount of Czech–English bilingual content, and we have managed to extract 801, 116 Czech and 5, 931, 091 English paragraphs from these domains. The extracted paragraphs were aligned with our method, creating a paragraph-aligned parallel corpus containing 114, 771 pairs from 2, 178 domains, having in total 7, 235, 908 Czech and 8, 369, 870 English tokens. The quality of the acquired corpus has been evaluated manually on a set of 500 randomly selected pairs. The precision was estimated to be 94, 60%. To evaluate the recall, we have selected one web domain (`www.csa.cz`) with a smaller number of paragraphs present in the input dataset. The results for the domain were estimated to be 95.45% recall at 97.67% precision.

We were surprised by the size of the corpus created in the second experiment, as we have expected to extract larger quantities of parallel paragraphs. However, we are convinced that the size is not affected that much by inferior recall of our approach, but the fact that the dataset does not contain many Czech–English web pages. The size of the corpus is comparable with the amount of Czech–English parallel data acquired by the previous project, focused on mining the Common Crawl datasets described in the related work [36]. Additionally, our approach achieves a higher precision.

The datasets produced by Common Crawl usually contain only a small subset of all the pages available on a web domain at the crawling time. Therefore, the approach described in the second experiment could be extended. We could use the list of web domains with identified Czech–English content to run our own targeted crawling that would navigate through the pages in more depth.

# Future Work

Both experiments show satisfactory results, implying that the proposed method is a promising baseline for acquiring parallel corpora from the web. Nevertheless, there is still some room for improvement. First of all, our method does not consider word order in the aligning process at any stage. Both the scoring function and the features designed for the classification could be extended to take this aspect into account.

Then there is the asymmetric nature of our method, meaning it generates different results if the source and the target languages are swapped. It could be extended to run the alignment for both directions and the results could be symmetrized. This might help the method achieve an even higher precision.

Finally, we have run our method only in a single-node environment so far. This is largely because we were aligning a relatively small sets of documents (not more than $15,000,000$). However, the method is designed to run in distributed fashion. Bins with input documents represent independent isolable tasks. Once the method is trained, these tasks could be distributed across multiple cluster nodes together with the resources needed for the aligning process. This would increase the throughput of our method and hence decrease the execution time.

# Bibliography

[1] Philipp Koehn *Statistical Machine Translation*, Cambridge University Press, 2009

[2] Jörg Tiedemann, 'Parallel Data, Tools and Interfaces in OPUS' in *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, European Language Resources Association (ELRA), 2012

[3] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean 'Efficient Estimation of Word Representations in Vector Space' in *In Proceedings of Workshop at ICLR*, 2013, `http://arxiv.org/abs/1301.3781` (accessed April 6, 2016)

[4] Tomas Mikolov, Quoc V. Le, Ilya Sutskever 'Exploiting Similarities among Languages for Machine Translation' in *Computing Research Repository, arXiv*, 2013, `http://arxiv.org/abs/1309.4168` (accessed April 6, 2016)

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean 'Distributed Representations of Words and Phrases and their Compositionality' in *Computing Research Repository, arXiv*, 2013, `http://arxiv.org/abs/1310.4546` (accessed April 6, 2016)

[6] Moses S. Charikar 'Similarity estimation techniques from rounding algorithms' in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, ACM New York, NY, USA, 2002

[7] Alexandr Andoni, Piotr Indyk 'Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions' in *Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions*, pages 117–122, ACM New York, NY, USA, 2002

[8] Jörg Tiedemann, 'Bitext alignment' in *Synthesis Lectures on Human Language Technologies*, pages 1–165, Morgan & Claypool Publishers, 2011

[9] Ondřej Bojar 'Čeština a strojový překlad: Strojový překlad našincům, našinci strojovému překladu', Institute of Formal and Applied Linguistics, Charles University in Prague, Czech Republic, Faculty of Mathematics and Physics, 2015

[10] Ondřej Bojar, Zdeněk Žabokrtský, Ondřej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jiří Maršík, Michal Novák, Martin Popel, Aleš Tamchyna 'The joy of parallelism with CzEng 1.0' in *Proceedings of*

*the 8th International Conference on Language Resources and Evaluation (LREC'12)*, European Language Resources Association (ELRA), 2012

[11] Common Crawl Foundation 'Common Crawl' `http://commoncrawl.org/` (accessed March 19, 2016)

[12] Miquel Esplà-Gomis, Mikel L. Forcada 'Bitextor, a free/open-source software to harvest translation memories from multilingual websites' in *Proceedings of the workshop Beyond Translation Memories: New Tools for Translators MT*, Association for Machine Translation in the Americas, 2009

[13] Miquel Esplà-Gomis, Mikel L. Forcada 'Combining Content-Based and URL-Based Heuristics to Harvest Aligned Bitexts from Multilingual Sites with Bitextor' in *The Prague Bulletin of Mathematical Linguistics*, volume 93, pages 77–86, 2010

[14] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer 'HTML5 – A vocabulary and associated APIs for HTML and XHTML', October 2014, `http://www.w3.org/TR/2014/REC-html5-20141028/` (accessed March 27, 2016)

[15] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau 'Extensible Markup Language (XML) 1.0 (5th Edition)', November 2008, `https://www.w3.org/TR/2006/REC-xml-20060816/` (accessed March 27, 2016)

[16] Yves Savourel 'TMX 1.4b Specification', June 2011, `https://www.gala-global.org/tmx-14b` (accessed March 27, 2016)

[17] Steven Pemberton et. al. 'XHTML$^{\text{TM}}$ 1.0 The Extensible HyperText Markup Language (2nd Edition)' January 2000, `https://www.w3.org/TR/2002/REC-xhtml1-20020801` (accessed March 27, 2016)

[18] Marco Lui, Timothy Baldwin 'langid.py: an off-the-shelf language identification tool' in *Proceedings of the ACL 2012 System Demonstrations*, pages 25–30, 2012

[19] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, Robert L. Mercer 'The mathematics of statistical machine translation: parameter estimation' in *Computational Linguistics - Special issue on using large corpora: II*, volume 19, issue 2, June 1993 pages 263–311, MIT Press Cambridge, MA, USA, 1993

[20] Dániel Varga, László Németh, Péter Halácsy, András Kornai, Viktor Trón, Viktor Nagy 'Parallel corpora for medium density languages' in *In Proceedings of the RANLP 2005*, pages 590–596, 2005

[21] William A. Gale, Kenneth W. Church 'A program for aligning sentences in bilingual corpora' in *Computational Linguistics - Special issue on using large corpora: I*, volume 19, issue 1, March 1993, pages 75–102, MIT Press Cambridge, MA, USA, 1993

[22] Iñaki San Vicente, Iker Manterola 'PaCo2: A Fully Automated tool for gathering Parallel Corpora from the Web' in *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'12)*, European Language Resources Association (ELRA), 2012

[23] Philip Resnik, Noah A. Smith 'The Web as a parallel corpus' in *Computational Linguistics - Special issue on web as corpus*, volume 29, issue 3, September 2003, pages 349–380, MIT Press Cambridge, MA, USA, 2003

[24] David Nadeau, George Foster 'Real-time identification of parallel texts from bilingual newsfeed' in *CLINE 2004, Computational Linguistics in the North East*, 2004

[25] John W. Ratcliff, David Metzener 'Pattern Matching: The Gestalt Approach' in *Dr. Dobb's Journal*, volume 7, page 46, 1988

[26] Philip Resnik 'Mining the Web for Bilingual Text' in *Proceedings of the 37th annual meeting of the Association for Computational Linguistics (ACL)*, pages 527–534, University of Maryland, College Park, Maryland, 1999

[27] Philip Resnik 'Parallel Strands: A Preliminary Investigation into Mining the Web for Bilingual Text', in *In 3rd Conference of the Association for Machine Translation in the Americas*, pages 72–82, Springer, 1998

[28] James W. Hunt, Malcolm D. McIlroy 'An Algorithm for Differential File Comparison' in *Technical Memorandum 75-1271-11*, Bell Laboratories, 1975

[29] Karl Pearson 'Notes on regression and inheritance in the case of two parents' in *Proceedings of the Royal Society of London*, volume 58, pages 240–242, 1895

[30] Krzysztof Wołk, Krzysztof Marasek 'Building subject-aligned comparable corpora and mining it for truly parallel sentence pairs' in *Procedia Technology*, volume 18, pages 126–132, International workshop on Innovations in Information and Communication Science and Technology (IICST), 2014

[31] George A. Miller 'WordNet: A Lexical Database for English' in *Communications of the ACM*, volume 38, pages 39–41, 1995

[32] Christiane Fellbaum 'WordNet: An Electronic Lexical Database', MIT Press, 1998

[33] Edward Loper, Steven Bird 'NLTK: The Natural Language Toolkit' in *Proceedings of the ACL 2012 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, volume 1, pages 63–70, 2002

[34] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, Evan Herbst 'Moses: open source toolkit for statistical machine translation' in *Proceedings*

*of the ACL 2012 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, volume 1, pages 63–70, 2002

[35] Philipp Koehn 'MOSES User Manual and Code Guide' 2016, `http://www.statmt.org/moses/manual/manual.pdf` (accessed April 5, 2016)

[36] Jason R. Smith, Herve Saint-Amand, Magdalena Plamada, Philipp Koehn, Chris Callison-Burch, Adam Lopez 'Dirt Cheap Web-Scale Parallel Text from the Common Crawl' in *Proceedings of the 37th annual meeting of the Association for Computational Linguistics (ACL) on Interactive Poster and Demonstration Sessions*, Pages 177–180 , Association for Computational Linguistics, 2007

[37] Jeffrey Dean, Sanjay Ghemawat 'MapReduce: simplified data processing on large clusters' in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation* volume 6, pages 137–149, USENIX Association, 2004

[38] Chris Callison-Burch, Philipp Koehn, Christof Monz, Omar F. Zaidan 'Findings of the 2011 workshop on statistical machine translation' in *Proceedings of the 6th Workshop on Statistical Machine Translation*, pages 22–64, Association for Computational Linguistics, 2011

[39] Holubová Irena, Kosek Jiří, Minařík Karel, Novák David 'Big Data a NoSQL databáze', Grada Publishing, a.s., 2015

[40] Straková Jana, Straka Milan and Hajič Jan 'Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition' in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Association for Computational Linguistics, 2014

[41] Drahomíra Spoustová, Jan Hajič, Jan Raab, Miroslav Spousta 'Semi-Supervised Training for the Averaged Perceptron POS Tagger' in *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 763–771, Association for Computational Linguistics, 2009

[42] Straka Milan, Straková Jana 'Czech Models (MorfFlex CZ + PDT) for MorphoDiTa', 2013, LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague, `http://hdl.handle.net/11858/00-097C-0000-0023-68D8-1` (accessed April 7, 2016)

[43] Straka Milan, Straková Jana 'English Models (Morphium + WSJ) for MorphoDiTa', LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague, 2014, `http://hdl.handle.net/11858/00-097C-0000-0023-68D9-0` (accessed April 7, 2016)

[44] Marcin Junczys-Dowmunt, Arkadiusz Szał 'SyMGiza++: A Tool for Parallel Computation of Symmetrized Word Alignment Models' in *Proceedings of the 5th International Multiconference on Computer Science and Information Technology*, pages 397–401, 2010

[45] 'HMM-based word alignment in statistical translation' in *Proceedings of the 16th conference on Computational linguistics - Volume 2*, pages 836–841, Association for Computational Linguistics, 1996

[46] Franz Josef Och, Hermann Ney 'A systematic comparison of various statistical alignment models' in *Computational Linguistics*, volume 29, number 1, pages 19–51, 2003

[47] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin 'Maximum Likelihood from Incomplete Data via the EM Algorithm' in *Journal of the Royal Statistcial Society*, series B, volume 39, number 1, pages 1–38, 1977

[48] Franz Josef Och 'An Efficient Method for Determining Bilingual Word Classes' in *Proceeding EACL '99 Proceedings of the 9th conference on European chapter of the Association for Computational Linguistics*, pages 71–76, Association for Computational Linguistics, 1999

[49] Qin Gao, Stephan Vogel 'Parallel implementations of word alignment tool' in *SETQA-NLP '08 Software Engineering, Testing, and Quality Assurance for Natural Language Processing* pages 49–57, 2008

[50] Chris Dyer, Victor Chahuneau and Noah A. Smith 'A simple, fast, and effective reparameterization of IBM model 2' in *Proceedings of NAACL-HLT 2013*, pages 644–648, 2013

[51] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning 'Bilingual Word Representations with Monolingual Quality in Mind' in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 151-–159, 2015

[52] Xin Rong 'word2vec Parameter Learning Explained' in *Computing Research Repository, arXiv* 2014, `http://arxiv.org/abs/1411.2738` (accessed April 6, 2016)

[53] Jeffrey Pennington, Richard Socher, Christopher D. Manning 'GloVe: Global Vectors for Word Representation' in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Association for Computational Linguistics, 2014

[54] Stephan Gouws, Yoshua Bengio, Greg Corrado 'BilBOWA: Fast Bilingual Distributed Representations without Word Alignments' in *Computing Research Repository, arXiv* 2014, `http://arxiv.org/abs/1410.2455` (accessed April 7, 2016)

[55] Erik Bernhardsson et al. 'Annoy: Approximate Nearest Neighbors in C++/Python', 2016, `https://github.com/spotify/annoy` (accessed April 11, 2016)

[56] Erik Bernhardsson et al. 'Benchmarks of approximate nearest neighbor libraries in Python', 2016, `https://github.com/erikbern/ann-benchmarks` (accessed April 11, 2016)

[57] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, Jürgen Schmidhuber 'PyBrain' in *The Journal of Machine Learning Research*, volume 11, pages 743–746, 2010

[58] Jiri Sima, Roman Neruda 'Theoretical Issues of Neural Networks', Matfyz-Press, Prague, 1996

[59] David E. Rumelhart, Geoffrey E. Hinton, Ronald J.Williams 'Learning internal representations by error propagation', in *Parallel distributed processing: explorations in the microstructure of cognition*, volume 1, pages 348–362, MIT Press, Cambridge MA, 1986

[60] 'ISO 28500:2009 - Information and documentation – WARC file format', International Organization for Standardization, Geneva, Switzerland. 2009, `http://www.iso.org/iso/catalogue_detail.htm?csnumber=44717` (accessed April 8, 2016)

[61] Timothy W. Bray 'The JavaScript Object Notation (JSON) Data Interchange Format', March 2014, `https://tools.ietf.org/html/rfc7159` (accessed April 18, 2016)

[62] Tom White 'Hadoop: The Definitive Guide, 4th Edition', O'Reilly Media, 2015

[63] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, Alex Buckley 'The Java Language Specification, Java SE 8 Edition', Addison-Wesley Professional, 2014

[64] Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley 'The Java Virtual Machine Specification, Java SE 8 Edition', Addison-Wesley Professional, 2014

[65] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung 'The Google File System' in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29–43, ACM New York, NY, USA, 2003,

[66] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler 'The Hadoop Distributed File System' in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010, IEEE Computer Society

[67] Apache Software Foundation 'HDFS Architecture', 2016 `https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html` (accessed April 18, 2016),

[68] Apache Software Foundation 'MapReduce Tutorial', 2016 `https://hadoop.apache.org/docs/r2.7.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html` (accessed April 22, 2016),

[69] Martin Kleppmann 'warc-hadoop: WARC Input and Output Formats for Hadoop', 2014, `https://github.com/ept/warc-hadoop` (accessed April 10, 2016),

[70] Jonathan Hedley et al. 'jsoup: Java HTML parser', 2015, `https://jsoup.org/` (accessed April 10, 2016),

[71] Anne van Kesteren, Aryeh Gregor, Ms2ger, Alex Russell, Robin Berjon 'W3C DOM4', November 2015, `https://www.w3.org/TR/2015/REC-dom-20151119/` (accessed April 10, 2016)

[72] Fabian Kessler et al. 'language-detector: Language Detection Library for Java', `https://github.com/optimaize/language-detector` (accessed April 10, 2016)

[73] William B. Cavnar , John M. Trenkle 'N-Gram-Based Text Categorization' in *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 161–175, 1994

[74] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay 'Scikit-learn: Machine learning in Python' in *The Journal of Machine Learning Research*, volume 12, pages 2825–2830, 2011

# List of Figures

# List of Tables

# Appendix A

# CD-ROM Contents

The contents of the companion CD-ROM are as follows:

- `Thesis/` the thesis in PDF format and its zipped LaTeX source code;
- `Output/` the acquired parallel corpus and the experiments' results;
- `CommonCrawl/` utilities for processing of the CommonCrawl dataset;
- `CzEng/` utilities for processing of the CzEng 1.0 dataset;
- `Tools/` tools and resources required by the method;
- `Align/` set of scripts implementing the method;
- `README.txt` manual for repeating the experiments;
- `LICENSE.txt` Apache License, Version 2.0.