DISSERTATION / DOCTORAL THESIS

Titel der Dissertation /Title of the Doctoral Thesis

„Automated Creation of Domain-Specific Bilingual Corpora for Machine Translation, focusing on Dissimilar Language Pairs“

verfasst von / submitted by

Bartholomäus Wloka MSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Philosophie (Dr.phil.)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 792 323
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Transkulturelle Kommunikation
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Gerhard Budin

To my father
I cherish his memory. His last words are with me, always.

To my soulmate
Sometimes the greatest acts of love are the hardest to commit.

Dla mojego Ojca.
Ostatnie jego słowa zawsze będą ze mną.

Dla mojej przyjaznej duszy.
Niekiedy te największe dowody miłości są te najtrudniejsze do wykonania.

Acknowledgments

I sincerely thank my advisor Prof. Gerhard Budin for his patience with my chaotic work style and the understanding he had for the turbulence that my private life caused to the work and progress of this thesis.

My heartfelt gratitude goes to Prof. Werner Winiwarter, with whom I spent countless hours discussing research, who always motivated me by encouragement and example, and time and again helped me in the process of shaping my professional and private life for the better. I thank him for critically and meticulously checking my work, and by doing so, helping me to improve and learn. For this, he sacrificed much time and effort even if it was often not officially recognized.

I thank my colleague Yasuko Yamamoto, who played a vital role in the evaluation of my results with her expertise in professional translation. I also thank my other former and current colleagues at the Centre for Translation Studies, Xiaochun Zhang, Vesna Lusicky, Barbara Heinisch, and Melanie Seltmann, for stimulating discussions, giving me additional motivation in my research, and their support in project work, organising and managing lectures and exams, freeing valuable time, which I could dedicate to this thesis.

Last but certainly not least, I thank my mother, who never missed an opportunity to lighten my day with a smile and an offer to help, in her self-less and honest way. And with all my heart I thank my partner; with her tough, resolute shell and her soft, loving core she helped me through very difficult times.

Contents

List of Figures

List of Tables

List of Codes

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BLEU	Bilingual Evaluation Understudy
CAT	Computer Assisted Translation
CPU	Central Processing Unit
DL	Deep Learning
GPU	Graphical Processing Unit
HDD	Hard Disk Drive
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
MT	Machine Translation
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NMT	Neural Machine Translation
PoS	Part-of-Speech
RBMT	Rule-Based Machine Translation
re	regular expression
SMT	Statistical Machine Translation
ST	Source Text
TT	Target Text
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

1 Introduction

1.1 Overview – Research Question and Output of the Thesis

The goal of this thesis is to provide a step towards the answer to the question: How can we automatically and efficiently create or extract high quality bilingual/parallel corpora, from freely available digital sources? Can we do this selectively for well-defined domains? Is it possible to find a method that can be used for any language pair without sacrificing quality? While parts of these questions have been answered, or are addressed in research with promising progress, combinations of these challenges taken together become a daunting task, especially if we consider dissimilar and low-resource languages.

In this thesis, we attempt to move closer towards a solution by addressing some of these questions. We present a method which extracts text in Japanese and in English from Wikipedia, based on a seed of topics. It aligns these sentences based on a novel metric we propose in this thesis. This method is transparent and traceable and can be adjusted and fine-tuned as needed. We present an implementation of this method in the form of a framework describing it in detail and with examples. The output, which we produced with this software, is evaluated by a translation professional and allows us to conclude with several empirical observations.

There are many multilingual digital sources available on the web. Amongst these many possibilities we chose Wikipedia for several reasons: One of them is simplicity. Wikipedia is well structured, so the extraction of text is straight forward and the semantic design of the content makes access, data acquisition, and cleaning of the data fairly easy. Further, Wikipedia is dynamic, it is constantly updated, hence repeated iterations, even with the same seed data, potentially yield new text. It is also highly multi-lingual and although the volume depends much on the particular language, there is a fair amount of content in most languages even in low-resourced languages and language pairs that are otherwise underrepresented digitally.

One particularly difficult question is: From all articles about one topic in several languages, how many are actually translated, how many are written in parallel and how many are composed independently. In other words, how much is parallel text, how much is merely a bilingual representation of the knowledge, and how much is an independent description. Examining this manually, although very time intensive, is very interesting in itself and fairly straight forward. Making judgements in such detail in a pool of thousands or millions of collected sentences obviously becomes impossible. Looking for translation candidates while not being aware of their availability and viability adds another level of complexity to the search for parallel data candidates. We decrease this complexity by using a transparent method to select the Wikipedia topics for extraction.

Similar to the manual evaluation of multilingual Wikipedia content, if few articles are selected and examined, we can make certain observations regarding the volume of text which is translated or equivalent. Our goal, however, is to collect as much data as possible. We attempt to strike a balance between collected volume, computational efficiency, and a transparent and traceable method.

Having addressed the choice of data source, and the selection of data within it, there is the question of the language pair. As already mentioned in the title we focus on English-Japanese. We think that this language pair provides a good case study for our approach as it meets our requirement of being dissimilar, i.e. differing significantly in surface characteristics, sociolinguistic factors, and cultural aspects as well as type of script. It is sufficiently represented on Wikipedia to offer a sizable overall volume of parallel corpora candidates. At the same time, this pair of languages still remains somehow underrepresented compared to the digital parallel resources of other widely spoken language pairs. In a time where the availability and quality of parallel data dictates the success of machine translation systems we think it is important to investigate any alternative for high quality parallel data acquisition.

The combination of all the above mentioned issues makes the problem very complex in its entirety. With a *divide and conquer* approach in mind we try to observe these closely interconnected questions separately:

- Can Wikipedia entries be considered as a reliable source for high quality English-Japanese parallel corpora?
- How much of English-Japanese Wikipedia content is a direct translation, how much is created in parallel, how much is created independently?
- How can we automatically obtain English-Japanese bilingual content from Wikipedia in an efficient manner?
- What is the most feasible way to automatically assess English-Japanese content crawled from Wikipedia?

To answer these questions we propose a method, explain our assumptions and premises, and present a proof of concept in the form of a modular framework. Our conclusions are based on empirical trials of collecting data with our software.

Our framework produces collections of bilingual text which is aligned, analyzed, and interpreted. We start with crawling Wikipedia for bilingual content. The type of content is defined by choosing seed topics. The algorithm proceeds to collect the text in both languages until a pre-set number of related articles is met. The text collection is then aligned, and each aligned sentence pair is assigned a score. This metric is a measurement of the equivalence between the Japanese and the English sentence. If the score is high enough, we deem this sentence to be a parallel data candidate. The output of this empirical portion of our thesis is:

- A modular software chain to automatically create parallel English-Japanese corpora.
- A sentence-aligned English-Japanese corpus with alignment scores for each sentence pair.
- An evaluation of the highest scoring parallel sentence candidates by a human expert.

1.1.1 Challenges

In this section, we expand upon the challenges which are mentioned in the introduction of this chapter. First of all, Wikipedia articles in two languages are not always translations of each other; quite on the contrary, the content often differs greatly depending on the language pair and the topic domain. Wikipedia article pages are often created independently in different languages. Sometimes they are created in parallel as a rough copy by paraphrasing parts of articles or sections. Occasionally, they are translated closely with attention to good language and detail.

This uncertainty whether the content is translated adhering to the original, translated by loose paraphrasing, independently composed, or a mixture of these three, makes the assessment very difficult. Additionally, Wikipedia content varies by many other aspects, such as culture, socio-political factors, technological advancements and simply the interest of the language representatives creating the content. A very good example of this variation is the Wikipedia entry for *Judo*; a traditional Japanese martial art. It is widely popular in Japan and is practiced from an early age in high schools, in clubs, and even as part of the physical education program. Although Judo has become an Olympic sport and is well known in the English speaking world, the percentage of practitioners and the media exposure is very small in comparison. This is very much reflected in the volume of the Wikipedia articles related to this topic with the Japanese entry currently covering roughly over ten times as much information. Such asymmetric volume and detail of articles is encountered more often if the languages are also separated by cultural, social, and historical differences. This phenomenon has a significant effect on the availability of parallel data depending on the domain. This has to be considered when choosing the seed topics and evaluating the collected parallel data.

Another critical issue is computational efficiency. Related to the topic of asymmetry discussed in the previous paragraph, if the selection of sections within articles is not done carefully most of the data will not be viable as parallel candidates. Such asymmetric data lowers the quality of the output and highly increases processing time. Sentence alignment is the most computationally intensive step in our framework. Looking at two sentences, one in English and one in Japanese, we have to compare each token of the first sentence with each token of the second sentence and determine whether there is equivalency. These comparisons alone result in an exponential runtime. Additionally to that, finding equivalencies is much more complex than one to one comparisons, due to polysemy, inflection, and other linguistic phenomena. This requires pre-processing, such as part-of-speech tagging, word-sense disambiguation and other methods which increase the runtime even further.

Hence, crawling large volumes of data without having at least an indication of a potentially high yield in terms of parallel data, i.e. without a certain preselection, would not be computationally efficient. Even the simplest way of preselecting material can make the alignment process more streamlined and open up computational power for more likely alignment candidates.

Last but not least, there is the challenge of judging the output of the collected and aligned data. Usually translation quality evaluation or translation post-editing is done with the assumption that the source and target sentence are indeed translations. This assumption cannot be made in the case of data that has been crawled from Wikipedia. This uncertainty of how much of the text is actually translated and how much was written independently or paraphrased makes evaluation even more challenging. Once we obtain this bilingual data, how can we assess translation quality, or whether they can be seen as translations at all? It is obvious that examining every piece of data in a dataset of thousands or even millions of entries by a human expert would not be possible, so a preselection in the form of an automated evaluation or a metric is necessary. An automatically computed metric during the alignment process makes a human expert evaluation feasible by selecting a manageable subset of data for evaluation.

A recent find (August 2020) has once more highlighted the importance of expert evaluations of Wikipedia content. A teenager from the United States has been authoring articles in the Scots language for years without being able to speak the language. This resulted in 27,000 Wikipedia entries which were linguistically wrong and misleading. It is astonishing that such a large number of wrong entries has gone unnoticed for years, until a perceptive reader finally pointed out the issue on social media. Scots, being an endangered and very low resourced language, is especially sensitive to being wrongly documented. The impact is especially critical on a language with low volumes of digital representation; but careless entries that go unchecked can have negative effects on data quality even for data rich languages, not to mention readers of Wikipedia being misled into making mistakes by faulty entries.

The conclusion is clear: black box, large scale data harvesting without quality checks is dangerous, in terms of quality and correct language. Good quality can only be obtained by transparent processes and human expert evaluation of at least a sample of the collected texts.

1.1.2 Approaches and Solution

In order to address the challenges mentioned in the previous section a method for preselection of topics is developed, so that bilingual content is not obtained randomly, but chosen according to link distance on Wikipedia.

For the assessment of the aligned sentence pairs a metric is devised and a score is assigned to each aligned sentence pair. To further examine the quality of the parallel data, a subset of sentence pairs with the highest score is evaluated by a professional translator.

1.1.3 Language-Specific Issues

Several intricacies have to be considered when collecting parallel data for Japanese and a Germanic language such as English. The issues presented here are specific but not exclusive to this language combination. We have tried to be as generic as possible in our description and our framework regarding the language pair; however, it is important to keep in mind that many of the issues are specific to English-Japanese which is our language pair of choice.

Zero anaphora constructions are very common in Japanese and often cause problems in translation and in this case, alignment. What makes them even more difficult is the fact that they often do not refer back to a previous part of the text, but rather to context, such as situation, tone as well as number, gender, age and social standing of people involved. The example in Table 1.1 shows merely a selection of possible translations of a Japanese phrase involving an assumption of a personal pronoun. Finding the right translation in the given context would require an analysis of the semantic proximity. This could be a neighboring sentence or even the entire paragraph.

Tokenization can be a problem in Japanese, even though the use of the Hiragana and Katakana syllabary for various grammatical and semantic functions often helps to encapsulate multi-character expressions. This multi script

Japanese	English translation
食べます	to eat
食べます	I eat.
食べます	You eat.
食べます	I am going to eat.
食べます	They will eat. (and many other possibilities)

Table 1.1: Translation polysemy examples.

property makes tokenisation easier compared to other Asian languages, such as Chinese, but it creates other intricacies, which are described later in this section.

Although there are many **polysemic expressions** in Japanese, a large number of them are homophones, hence do not pose a significant difficulty when they are examined in written form. The words in Table 1.2 are examples of this concept. Each word is pronounced "kōshō". The difficulty in this case is limited to finding the right form according to the characters.

Grammatical **structural differences** are a considerable issue. Comparison of equivalent expressions on a phrase level is very difficult between English and Japanese, due to the auxiliary verb and particle structure in Japanese. Although there are equivalents in English, finding such structures using an al-

Hiragana	Kanji	English translation
こうしょう	交渉	negotiation
こうしょう	公証	authentication
こうしょう	考証	historical investigation
こうしょう	校章	school badge
こうしょう	高尚	noble
こうしょう	公称	nominal

Table 1.2: Phonetic polysemy examples.

Japanese	English translation
これは私のメガネです。	These are my glasses.
次の電車に乗る	Take the next train.

Table 1.3: Example of particle usage.

gorithm is not reliable, since they often do not follow a well defined structure. The example shown in Table 1.3 demonstrates the different ways of using the "no" particle in Japanese, which usually indicates possession or belonging to something and is often compared to a genitive case in English. Equating the particle "no" with a genitive case every time would result in many mistakes. Thus, the lack of definite equivalents for particles makes the comparison of phrases unreliable and costly.

Japanese has adopted many words from English and other European lan-

Japanese phoneme	possible transcriptions
フ	fu/hu
リ	ri/li
レ	re/le
ブ	wu/bu

Table 1.4: The Japanese phonemes are written in the Katakana syllabary. Both transcriptions and pronunciations on the right side are possible.

guages, especially when it comes to modern terminology. These terms, however, are written in the Japanese script. **Transcription** of these words is not always a trivial task. These borrowed words are usually written in the Katakana syllabary and are sometimes abbreviated to the point of obscurity. Katakana is built on a different set of phonemes, therefore a transcription quite often results in slightly changed words. Japanese has fewer phonemes than English, and often two different phonemes are transcribed into one representation. This potentially leads to a situation where a transcription back to English yields a different word. Quite often an adopted foreign term is changed to such a degree that it makes more sense to consider it a Japanese word, rather than a transcribed adoption of a word. See Table 1.4 for a few examples of this phenomenon.

Such transcription problems can potentially result in rather amusing consequences when, for example, encountered on a menu in a restaurant, as seen in Figure 1.1.

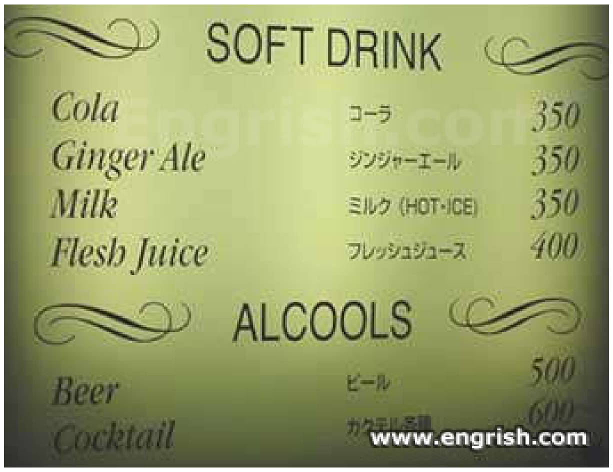

SOFT DRINK
Cola コーラ 350
Ginger Ale ジンジャーエール 350
Milk ミルク (HOT·ICE) 350
Flesh Juice フレッシュジュース 400
ALCOOLS
Beer ビール 500
Cocktail 600
www.engrish.com

Figure 1.1: Unfortunate transcription errors on a restaurant menu. (source: www.engrish.com)

Character encoding can be a significant problem, especially when reading and writing Japanese characters to memory or into files. There are several encoding systems used in Japan. The common denominator between digitally representing Japanese and languages based on the Latin alphabet is usually *utf-8*, however, decoding and encoding is often a tedious task inviting mistakes. Many Japanese language resources are only available in certain encoding formats, which requires even more additional conversions.

1.1.4 Objective and Scope

To summarize the previous sections in this chapter, the key objectives of this doctoral thesis are:

- An algorithm to efficiently obtain corresponding text from English and Japanese Wikipedia pages.
- An algorithm to align the English-Japanese sentences pair candidates.
- A metric to quantify the similarity between an English and a Japanese sentence, i.e. signifying the likelihood of these sentences being a translation or giving a rough idea of equivalence.
- Implementation of the above mentioned algorithms and the metric in order to obtain a significant number of sentences.
- Assessment of a sample of the obtained parallel data by a human translator.

1.2 Structure of Thesis

The main body of the thesis is structured similarly to the modular nature of the framework which was developed as its proof of concept. We describe the stages, modules, and functions of this framework parallel to the theory that is behind it.

We start with a discussion about related work, the scientific background, and the motivation for this thesis in Chapter 2. We begin with a brief look at languages and translation in general, followed by machine translation and its most recent paradigm: neural machine translation. We introduce the topic of language data, in particular parallel corpora and their importance, and present a collection of existing resources. We conclude this chapter returning to the topic translation, albeit this time from the perspective of evaluation.

In Chapter 3 – Automated Wikipedia Corpus Acquisition Tool – AWCAT – we give an overview of the implementation of the software, which we developed as a proof of concept. We outline the architecture of the stages and modules in this chapter, and add the specifications of the hardware which was used in the course of developing and running the framework.

In Chapter 4 – Data Extraction– we talk about Wikipedia as a data source, how we select data for crawling; and we describe the Data Extraction Stage. We explain how we have created a glossary building functionality in a module in this stage.

Chapter 5 – Data Preparation – is dedicated to the cleaning and preparation process of the data. We describe the Data Preparation Stage which optimizes the format of the data for alignment. We also explain how we prepare the language resources which we use in the following stage.

In Chapter 6 – Sentence Alignment – we elaborate on how the bilingual text, which was obtained and formatted in the previous stages, is aligned to become a corpus of parallel sentence candidates. We start with a description of the algorithm, explain the alignment metric, break down a process of aligning into individual steps as an example and finally explain the inner workings of the Sentence Alignment Stage.

Chapter 7 – Evaluation – presents the various methods of assessment and ranking, i.e. the automatic evaluation with the metric from Section 6.2 and human expert evaluation.

In Chapter 8 – Conclusion – we summarize the results and discuss the findings. We share our observations from the process of developing the framework. We present the potential contributions of this thesis and describe what steps need to be taken to adopt the framework to other languages. We mention the limitations of our approach and our framework and finally conclude with an outlook on future work.

There is no data like more data.

Robert Leroy Mercer

2 Scientific Background

2.1 Languages and Translation

For centuries scholars and researchers have tried to describe languages. These endeavours were undertaken from many different perspectives, but one major common goal was to find the underlying concept of all languages, to discover patterns, common structures and similarities to eventually conquer the language barrier. It is apparent that language encodes not only what we can perceive with our senses, e.g. our physical surroundings, but also specific concepts, views and traditions and cultural idiosyncrasies. Even tangible things, such as forms of flora and fauna, which can differ greatly depending on the geographical location, define the properties of a language, ranging from terminology to specific idioms, similes and analogies. Many other factors changing dynamically according to the political, cultural, and technological development of its population make language extremely divers and complex. Nonetheless, there were many attempts to define a formal comprehensive representation of all languages, an *Interlingua*, a language independent representation of concepts, serving as a *hub* between languages to which every language could be converted and from which a conversion into any other language would be possible by applying analysis and construction according to rules. In theory this would have revolutionized machine translation, but it turned out to be impossible to fully realize.

With the dawn of the *Information Age* and the dramatic change of the way written word is processed, the perception of language has changed significantly. Just as Johannes Gutenberg revolutionized the spread of language and information by the invention of the printing press, so has the computer and the subsequent interconnectedness of computers throughout the world, the World Wide Web, revolutionized it once again. Thus, the perception of language was changed once more, this time on a global scale enabling virtually anyone to create and disseminate written content.

The digital representation of language with the electronic computer made it possible to program machines to translate between languages. This history of automated translation reaches back to the end of World War II. In the beginning stages, it was pursued with limited success, but soon was boosted with the availability of an ever growing volume of multilingual digital data. A short overview of the chronological development and a selection of associated research milestones of *Machine Translation* (MT) is described in the following section, followed by the newest paradigm in MT: *Neural Machine Translation* (NMT). Following this, we discuss the importance, availability, and research in language data. We explain why it is so important and why we rely so heavily on bilingual/parallel corpora in MT. We conclude this chapter with a discussion about translation quality evaluation.

2.2 Machine Translation

The decoding of information as it was done by Alan Turing in the 1940s on the first generation of programmable digital electronic computers has sparked the interest of the research community, and the idea to apply a similar method to transpose information from one language to another gave birth to *Machine Translation* (MT).

The evolution of the research field of MT has undergone many paradigm shifts since then, as described in (Hutchins, 2012) and (Arnold et al., 1993). Word to word translation systems, which utilized dictionaries to directly map words from source to target language, were the first methods used, taking advantage of the speed at which a computer could look up entries in a digital dictionary. Although such a translation offered a rough idea of the source text for some language pairs, the translation was often wrong due to polysemic words, hardly readable due to the lack of consideration of grammar and all but reliable.

More intricate and analytic methods have been researched since then. The next logical step after the direct word transfer approach was the analysis of grammatical rules. Rule-based transfer systems allowed for a better translation from the source text and a more coherent representation in the target language, such that the transposed content was easier to read and for the most part syntactically correct. The most difficult problem, however, was polysemy in both source and target language. The disambiguation of words and phrases has been a focus of research for decades and remains an issue until today. For instance, to successfully identify the meaning of the word *bank*, one has to observe the context to find whether a *river bank* is meant, or the financial institution. The transfer-based method was extended by the semantic analysis. In theory, this analysis of the sentence by rules and meaning would result in an *Interlingua*, a formal, language independent representation of the source text. This Interlingua could then be reconstructed into the target text by essentially reversing the analysis process. This *Transfer-based MT* approach is visualized in a simple and intuitive way by the Vauquois triangle (Vauquois, 1968), shown in Figure 2.1.

Another paradigm in MT, *Phrase-Based Machine Translation* and its most popular version *Statistical Machine Translation* (SMT) was introduced as early as 1949 by (Weaver, 1955) but did not take off until the 1980s, when it was made popular by IBM. It was around this time, that bilingual digital text data was starting to be available in significant quantities. The statistical analysis of large volumes of bilingual data results in probabilities for translating words

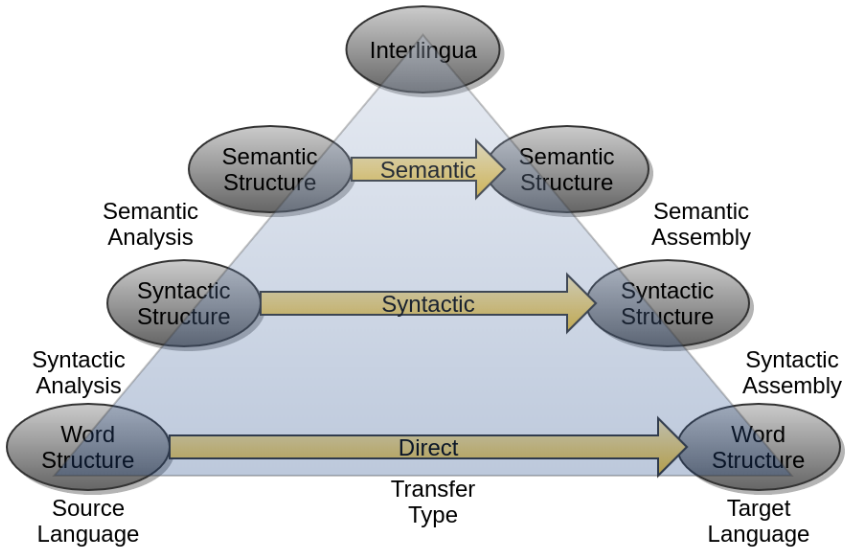


Figure 2.1: The Vauquois translation triangle depicting the possible transfer levels of the transfer-based machine translation approach.

and phrases without the machine having to analyse the semantic intricacies of the source text. The basis for the *training*, i.e. obtaining the probabilities of translation of words and phrases from source to target text, are *parallel* corpora. These are text corpora in two languages, where each sentence of the source language is aligned with the sentence of the target language. The higher this data is in quality and volume, the more accurate the probabilities of translations.

According to these probabilities obtained in the analysis step, the equivalents of words and phrases (so-called *n-grams*, where *n* stands for the number of words in the sequence) with the highest probability can be selected to construct translation candidates. This is often augmented by a language model, a probability distribution over sequences of words, which helps to improve the translations by correcting the word order in the target sentence.

In general, the larger the bilingual data set, the better the result, although one must take into account structure, domain, style, and general form of the data to best match the domain to which that system will be applied. Additionally, the SMT method is very well suited to be augmented with methods from other MT paradigms, such as rule-based systems, lexical lookups, terminology databases, distributional semantics, etc., to become a *hybrid* MT system.

The most recent paradigm of machine translation is *Neural Machine Translation* (NMT). This method is based on *deep learning* which has emerged in *Artificial Intelligence* (AI) research. Deep learning mimics our brain's network of neurons and has revolutionized AI in recent years (Koehn, 2017). The NMT paradigm had a significant impact on MT and the biggest providers of online translation services, such as Google and Microsoft switched to NMT several years ago. The history, technical background, implementation and the consequences it had on MT research are discussed in Section 2.3.

It is important to point out that even though the newest candidate, the NMT method, has taken most of the spotlight of current MT research, each of the above mentioned MT methods has been used successfully until today. The applications of these more traditional methods have specific advantages and very often *hybrid MT systems* are constructed; these customized solutions emphasize desired functionalities, exploit strengths, while minimizing the impact of potential weak points.

The requirements of a system arise from the language pair being translated, the text domain, the need for accuracy, the need for coverage, and many more aspects. A limiting factor is often the availability of corpora for a certain language pair and differences in surface characteristics between languages. This demands a careful selection and combination of MT methods. Data processing and analysis depth, use of lexical or semantic resources, rule-based, statistical, neural methods; all these aspects offer advantages, if combined sensibly for the desired application. For example, so-called lesser resourced languages benefit from other methods than languages which are rich in digital corpus data. The differences in surface characteristics for a language pair also pose challenges, therefore a careful selection of the MT method is decisive for a good result. Another important consideration is how the system is applied and the desired goal: Is it a fully automated system for a broad audience, or is it a *Computer Assisted Translation* (CAT) tool for translation professionals? Last but not least, there is the matter of the translation domain, such as legal, medical, technical, scientific, etc. The better the training data adheres to the desired domain, the better the result. Mercer's quote, which headlines this chapter, certainly benefits from this additional consideration, especially in the context of current development, where data is abundant, but information relatively scarce.

2.3 Neural Machine Translation

NMT, being currently the dominant paradigm in MT, deserves a separate mention in this thesis due to its importance as a data-driven paradigm. Although SMT also relies heavily on data and can be certainly described as data-driven, the need for even larger volumes of data makes NMT stand out.

The biggest difference between these two methods is probably that NMT is an end-to-end translation, meaning that the entire process starting from decomposition of the words and characters into an abstract digital representation which is processed in hidden layers, until the reconstruction into characters and words in the target language is a *black box* process. This process is depicted schematically in Figure 2.2. This was successfully presented in (Luong and Manning, 2016), showing promising results for highly-inflected lan-

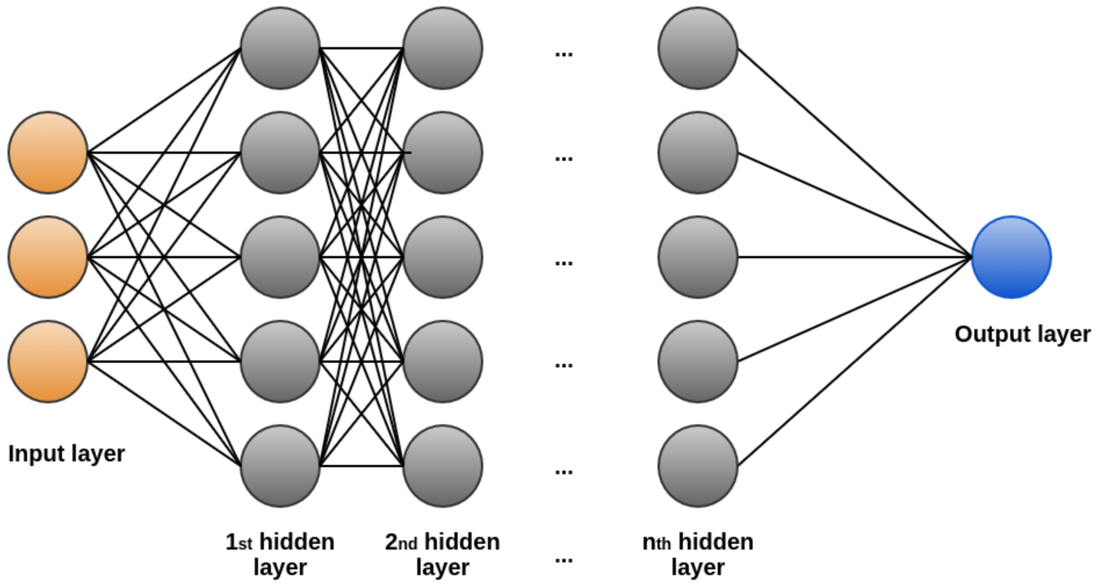


Figure 2.2: Schematic representation of an artificial neural network with n hidden layers.

guages with a very complex vocabulary.

The nodes in the *input layer* are connected in a way that represents the data we want to process. This could be any data, ranging from image pixels for pattern recognition in pictures to frequency values for voice recognition. In the case of MT this is text data, i.e. sentences, words, characters in the source language broken down into a binary representation.

In the first hidden layer these binary inputs are either conveyed further (1), or are not relayed (0). The decision whether one signal is relayed or not is dependent on the weight function, which was adjusted in the training process of the neural network. The output of the first layer is relayed into the next hidden layer, and so on. In the end the data is converted into characters, words and sentences in the target language.

It is obvious that these multitudes of connections and their weights, which are based on millions of adjustments during the training phase, cannot be retraced for one particular choice of word or phrase in the translation process. This means the decisions of the NMT systems cannot be logically justified, nor can any other method be added in a meaningful way to augment the quality of the translation, unless it is done before or after the neural network.

Consequently, the most crucial aspect of a NMT system is the training, during which the potentially millions of neurons of the hidden layers are assigned weights and adjusted continuously during the training process. This means that NMT systems require an exceptionally large volume of training data, and they cannot be supplemented easily with other paradigms.

Most recently, the *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2018), which is a pre-trained language model, based on neural network architecture, has been applied to NMT (Zhu et al., 2020). Although BERT was initially envisioned for fine-tuning of tasks like text understanding and language inference, the application of these models to NMT with the so-called BERT-fused model seems to yield promising results.

2.4 Language Data in Machine Translation

As described in Section 2.2 and Section 2.3 all paradigms of MT rely on data, albeit the data requirements are different. The most recent paradigms, SMT and especially NMT, require large volumes of parallel data and are highly dependent on their quality. Hence, regardless of which method is chosen, there is always a need for bilingual, and even better, parallel corpora.

Since the birth of the Internet and the boom of the World Wide Web, more and more text data is freely available, partially in bilingual or even multilingual form. Additionally, the steady increase of computational power and efficiency has made the processing of large volumes of text feasible.

This text data has been utilized to train MT systems and the results can be seen readily in popular MT systems like *Google Translate* or *Bing Microsoft Translator*. Despite the progress in utilizing large volumes of bilingual data, the process of collecting and preparing this data remains tedious and difficult.

Quality and domain adherence as well as the evaluation of large collections of text tend to be the most daunting tasks, being highly computationally intensive and laborious. Last but not least, understanding a text, with its connotations, general tone, and implications is in some cases even difficult for an expert. Hence translations often differ, depending on the translator, and quality of translation is therefore very difficult to judge, even for professionals.

Thus, collecting bilingual resources for the use in language technologies is a profound and intricate issue. Requirements for such corpora can differ greatly, depending on the context and application, but it is undoubtedly true that better quality *and* quantity are needed to achieve good results. This issue is further elaborated in the next section.

2.5 The Importance of Bilingual/Parallel Corpora

To translate a text, e.g. transpose a concept from one language to another is to carry over meaning, culture, emotions and associations into a different representation of meaning, culture and associations. With this premise we can safely say, at least for now, that fully automated high quality machine translation is not a realistic goal for all applications. Whether reliable, high quality, and accurate translation of all domains will be possible with artificial intelligence of the future remains a highly debated issue, but as of now, reliable translation results have been only achieved for well defined domains or controlled languages for fully automated systems. The key to success is good data in large volumes.

A machine translation will only be as good as the example it was given. Hence, whichever method we use for translation, we need example data to achieve our goal. Even in the case of a strict rule-based system, we derive these rules from the use of the language. In the case of SMT this is even more obvious and the volume of text data needs to be very large. In fact, the more data we use to train SMT the better the result tends to be. However, at some point, after including a huge amount of data, available to, e.g. Google or Microsoft, we seem to reach a certain saturation and occasionally a slight decrease in accuracy.

Mercer's general and also bold statement seems to be at danger here, although, if one considers that not all data is used for only one single system, but instead, segments of data are intelligently selected for different systems, depending on the application, one might argue that the statement still holds, albeit only under certain assumptions. Thus, it is vital to put emphasis on intelligent selection and distribution of the data at our disposal.

2.6 Bilingual/Parallel Corpus Acquisition

Cross-language text retrieval has been practiced since the early 1970s (Salton, 1971). It was not until the personal computer era and the global spread of the

World Wide Web, and with it the exponentially growing volume of textual data that this area became active on a large scale. Since then, the language technology community has been stressing the importance of high quality multilingual data, in particular for MT (Rapp, 1999; Kumano and Tokunaga, 2007; Zhao et al., 2008). SMT, which up until a few years ago was used by *Google Translate*, requires large volumes of parallel corpora to produce adequate output as established early in the SMT research (Koehn et al., 2007). The since then emerging new MT method called *Neural Machine Translation* (NMT) uses a different way to train, but requires an even larger volume of data, as described by (Koehn, 2017).

The first experiments in multilingual document retrieval in direct combination with MT have been performed since the late 90s (Braschler and Schäuble, 1998). Since then research efforts in this field have been increasing constantly, no doubt, driven by the quick rise of the world wide web and its multilingual nature.

Large scale projects have been undertaken to create a comprehensive solution to fulfill the quickly growing need for parallel corpora. There have been approaches to automatize the process of parallel corpus creation. The aligning process is by far the most difficult and time consuming step, so that methods were developed to both assist manual alignment (Grimes et al., 2012) as well as fully automatic alignment (Chen and Eisele, 2012; Cakmak et al., 2012). Furthermore, the importance of domain-specific corpora and hence domain-specific crawling has been addressed, e.g. in the touristic domain (Espla-Gomis et al., 2014).

A very efficient way to obtain large parallel data collections is to take advantage of existing translated texts from international organizations. This was done in the *Europarl* project (Koehn, 2005) where texts from the proceedings from the European Parliament were aligned to create large parallel resources. Another such example is the *United Nations Parallel Corpus* (Ziemski et al., 2016), a collection of consolidated and processed data from UN documents. These projects take texts translated by professionals, which makes aligning them fairly straightforward, especially since they are often annotated with rich metadata.

A much more difficult source to mine parallel text is the web, but it goes without saying that it is a bigger source and offers much more diverse data. A large scale distributed system for parallel text mining was proposed by (Uszkoreit et al., 2010), and (Smith et al., 2013) suggested a method for aligning sentences obtained from Common Crawl [*].

[*] `https://commoncrawl.org/` (Last accessed in August 2020.)

As the methods of obtaining parallel data improved and more language pairs were included, so did the need for parallel data. Data intensive NMT requires this data quickly on-demand, for any language pair, and for any domain. One example of a large scale effort which addresses this issue for European languages is the *ParaCrawl* project, which is co-financed by the European Union. The ParaCrawl group published their most recent findings just weeks before the time of writing this thesis and focuses on testing various existing alignment methods in their corpus creation software tool chain (Bañón et al., 2020). The methods included in these tests are *Hunalign*, *Bleualign*, and *Vecalign*.

The alignment with *Hunalign* requires a bilingual dictionary and is based on sentence length similarity and the IBM translation model (Brown et al., 1993), or alternatively on a dictionary-based model, enhanced by a language model in the target language. This algorithm was initially used for Hungarian, Romanian, Slovenian, and English. It was published by (Varga et al., 2005).

Bleualign uses the *BLEU* translation assessment score, which is discussed in Section 2.8 further down in this chapter. Instead of using BLEU as a quality measure for translation, the scores for all sentences are computed and the highest scoring sentence is assigned as the parallel sentence (Sennrich and Volk, 2010).

Vecalign (Thompson and Koehn, 2019) is the most recent alignment method and relies on *sentence embeddings*, which is also described in more detail in Section 2.8. The pre-trained embeddings used for ParaCrawl were *LASER* embeddings (Artetxe and Schwenk, 2019), which covered all ParaCrawl languages, except for Irish.

The results of the comparisons are too complex to be summarized in a few sentences and we recommend to study the result tables provided by (Bañón et al., 2020) for further details. In a nutshell, each approach had advantages and disadvantages, but the overall issue for all of them was the computational costs and the resulting energy use. On a side note, albeit on a very important one, (Bañón et al., 2020) makes this very progressive step to go beyond a discussion of computational cost, which tends to carry with it only the connotation of "slow" and "we need to make it faster". In a time of energy source debates and climate change we might want to start thinking about "wasteful" versus "energy efficient" instead. According to the publication, pre-processing 100TB of data to produce parallel corpus candidates consumed 50,000 CPU-hours, which requires an estimated 750kWh. This is more than a month of energy supply for a developed country average household. This makes computational time one of the biggest problems in automatic parallel corpus creation, apart from issues with quality and quantity.

This high cost in computation stems from the fact that some steps in the process have an *exponential runtime*. This means that the steps required to traverse n pieces of data is n^k, where k is some constant greater than 1. Hence the number of computational steps increases progressively, the more data is processed. One of the steps which contributes to this exponential runtime is the alignment, where each combination of words of both source and target sentence have to be checked. Often lookups of lexical entries have to be performed as well.

The most recent approaches with word embeddings, which can utilize the power of *Graphical Processing Units* (GPUs) remedy this problem to a certain degree. However, these approaches are used to process huge volumes of data. Some of this data might be irrelevant for the success of these computations and the achieved computational efficiency is thereby negated. It is always better to pre-select data, in order to be more efficient and to process only likely candidates or high quality sources. One example of such a focused crawling technique has been shown by (Laranjeira et al., 2014). It seems that these approaches have been partially forgotten, in favor of *Big Data*.

The methods mentioned so far have addressed fairly similar languages in terms of script, surface characteristics and sociolinguistic factors. Highly dissimilar languages, and especially language pairs with scarce availability of resources, pose an even greater challenge, in terms of identifying, processing, and aligning data. One of these research efforts, which focuses on alignment, is done by (Ma, 2006), where an approach of aligning English and Chinese sentences based on the frequency of words is presented. Recent trends favor approaches which address simultaneous harvesting and aligning, so parameters of harvesting can be adjusted to benefit alignment, and vice versa, resulting in a more efficient system overall. A recent example is (Aker et al., 2012), where the focus is efficiency of harvesting to ensure a high recall during data collection to result in less, but better data for aligning. This is achieved by finding potentially useful texts by title, together with time-stamps of the text which combined, is a very good indication of equivalence. However, this is obviously restricted to data where such information is readily available, such as news articles.

Inspired by the previously mentioned *ParaCrawl*, and addressing the still limited availability of Japanese-English parallel corpora, (Morishita et al., 2019) started the *JParaCrawl* project and used the methods from *ParaCrawl*, which we described before, to compile a Japanese-English corpus. This effort has resulted in a collection of 8.7 million sentence pairs. Just as *ParaCrawl*, the data collection is not divided into domain categories, but is a good base for generic NMT systems.

The authors provide a comparison of *JParaCrawl* pre-trained, domain-specific NMT models with additional Japanese-English corpora for fine-tuning (*ASPEC*, *JESC*, *KFTT*, *TED Talks*, see Section 2.7).

The *WikiMatrix* project certainly deserves a separate mention, especially since it used Wikipedia as its data source, as we do in this thesis. Parallel data in 85 languages has been extracted, amongst them even dialects and low-resource language pairs. Using the aforementioned LASER sentence embeddings (Artetxe and Schwenk, 2019), respectable BLEU scores were achieved for some of these languages. However, as can be seen in the tables provided by (Schwenk et al., 2019), some language pairs scored rather poorly. One of these poorly scoring pairs is Japanese-English, alongside other dissimilar language pairs. This confirms once more the shortcomings of the *Big Data* approach for these language pairs. Especially in these cases transparency would be very beneficial. It would help to identify whether the problem is the data, the alignment, or some other part of the process.

Exactly for that reason, neural network approaches are not always the best approach or at the very least would benefit from traceable processes, in order to make the path to the results more transparent. However, such transparency is nearly impossible. Nevertheless, approaches which collect text by taking advantage of deep learning models are increasingly popular. Some models yield adequate results even for low-resourced languages, following the *Zero-Shot* approach (Johnson et al., 2016). However, an important consideration is that if we artificially create data to make up for a resource gap, this artificially created data will become input during the next cycle of data harvest, and will be used to train the next generation of neural machine translation, which in turn may become input for the next cycle, and so on. Such recursive use of data can cause problems on many levels. Approaches which consider transparent methodologies should be considered before this feedback loop gets out of hand.

One additional benefit of building corpora with a transparent methodology is the application of parallel data for language learning purposes. Word similarities, equivalent terms, topics, and proper nouns, together with the sentences in which they are contained as examples, can all be used in a computer enhanced language learning environment. Such a combination of resources with language learning environments, which are integrated seamlessly into Web content browsing is shown by (Winiwarter, 2013, 2015). Information which is taken from the creation process of the parallel corpus with a traceable method can be used to enhance existing language learning platforms and create new applications for language learning and understanding; undoubtedly an important concept in a time where technology is starting to be increasingly *black box*.

2.7 Existing Parallel Corpora for Japanese/English

When it comes to parallel corpora the definition of the term "lesser resourced" is quite vague. Whether Japanese-English falls into this category is up for debate, but it is certainly true that the data currently available can be classified as limited at best. We compiled a list of of already available parallel corpora for this language pair. Table 2.1 shows the resources in alphabetical order. This list is by no means exhaustive, since there are many more company-owned, proprietary text collections. The focus of this compilation is an overview of openly available resources, which are free for scientific use and require at most an agreement or contract with the copyright holder or creator. Table 2.2 lists the URLs of these resources for the reader's convenience.

Resource name	Sentence count	Content/domain
ASPEC	3.0M	Scientific abstracts
JENAAD	150k	News articles
JESC – Japanese-English Subtitle Corpus	2.8M	Subtitles
JParaCrawl	8.7M	Generic web content
Kyoto Wiki (KFTT)	330k	Wikipedia articles
NTCIR PatentMT	3.2M	Patents
TED Talks	100k	Translated subtitles from TED talks
Tanaka Corpus	150k	Collected by language students

Table 2.1: Details of freely available English-Japanese parallel corpora.

Resource name	URL
ASPEC	http://lotus.kuee.kyoto-u.ac.jp/ASPEC/
JENAAD	(http://www.nict.go.jp/en/) not available as of August 2020
JESC – Japanese-English Subtitle Corpus	https://nlp.stanford.edu/projects/jesc/
JParaCrawl	http://www.kecl.ntt.co.jp/icl/lirg/jparacrawl/
Kyoto Wiki (KFTT)	http://www.phontron.com/kftt/
NTCIR PatentMT	http://ntcir.nii.ac.jp/PatentMT/
TED Talks	https://wit3.fbk.eu/
Tanaka Corpus	http://www.edrdg.org/wiki/index.php/Tanaka_Corpus

Table 2.2: URLs of freely available English-Japanese parallel corpora.

The first listed resource, the *ASPEC* corpus, is a large collection of aligned scientific abstracts by (Nakazawa et al., 2016). Since scientific publications of various research areas are covered the domain is not restricted by research field or topic, but rather by the general way of writing, namely a formal, descriptive and narrative way, generally used in research articles. In the scope of this dissertation, we used the *ASPEC* corpus as a reference to test our alignment metric in the initial stages of development. *JENAAD*, compiled by (Utiyama and Isahara, 2003), has been built from news articles, and is in that regard similar to *ASPEC* except that it covers the news domain. *JESC*, the Japanese English Subtitle Corpus by (Pryzant et al., 2018) is a database of subtitles which were crawled from movies and TV programs available on the Web. *JParaCrawl* is the largest and most recent in this collection and is created by (Morishita et al., 2019); it is discussed in more detail in Section 2.6. The *KFTT Kyoto Free Translation Task* is a corpus of Wikipedia articles related to Kyoto, which were manually checked. The *TED Talks* corpus, described by (Cettolo et al., 2012), is a translation of transcriptions and subtitles from TED events. The *Tanaka Corpus* is a collection of parallel sentences, collected by students, and was published by (Tanaka, 2001).

As mentioned earlier, this list does not contain all resources, but it can be undoubtedly stated that the availability of parallel corpora for the Japanese-English language pair for scientific research is far smaller than it is the case for major European language pairs, or even English-Chinese. For a comprehensive overview of available resources for these languages we refer to *ELRA Catalogue of Language Resources*[†], and the *LDC Catalog*. [‡] Furthermore, it is

[†]`http://catalogue.elra.info/en-us/` (Last accessed in August 2020.)
[‡]`https://catalog.ldc.upenn.edu` (Last accessed in August 2020.)

important to remember that languages are dynamic and different application scenarios demand domain-specific data, so availability of large heterogeneous datasets is vital, but the ability to quickly and efficiently build new resources is equally important.

2.8 Translation and Corpus Evaluation

One of the most important considerations while creating MT systems and parallel corpora is the evaluation of translation quality. While human expert evaluation is undoubtedly the most accurate and reliable method to judge translation quality, the volumes of data which need to be assessed often make a manual approach difficult and sometimes impossible. In a nutshell, quality assessment by experts is highly accurate but time consuming since tone, register, connotations, sociocultural nuances, and many other aspects are considered in a fine-grained analysis. Quality assessment of translation is an important chapter in translation science and has been researched for a long time. Automatic evaluation, which will be described in the following paragraphs, offers standardized results, and most importantly can process data in a matter of minutes or hours depending on the size of the data set.

Consequently, the method of evaluation depends on the requirements, practicality, and feasibility. Literature translation demands experts who apply a very precise, at the same time flexible and creative process, while the evaluation of user manual translations certainly does not require deep thought processes to decide their usefulness and accuracy. Obviously, the majority of the translated texts which need to be evaluated fall somewhere in between these two extremes of the spectrum. Often it is advisable to seek a compromise and to find methods which maximize the advantages from both sides. Human expert quality assessment will not be discussed in further detail, since this would infringe on a wide and involved research area in translation science, which is not the focus of this thesis. Automatic evaluation however requires a closer look for a better understanding of the thought process in this thesis.

The approach in automatic evaluation of translation comes from the perspective of computer science, a field much closer to mathematics than linguistics. This resulted in the attempt to create a quantifiable quality measure with a transparent and logical reasoning. Such a method is not meant to consider any of the complicated linguistic concepts – at least not explicitly – but relies purely on statistical similarity to reference data, i.e. corpora with previous translations. This automated method was found to be useful in providing quick, coarse estimates of translation quality.

One of these measures is BLEU by (Papineni et al., 2002) and the closely related NIST metric by (Doddington, 2002). BLEU quickly became the de-facto standard metric for evaluation of automatic translation. On one hand, this method has been widely used in the MT research community and is mentioned in many if not most scientific publications; on the other hand it has been also debated and criticised.

In order to understand this dichotomy, we need to understand what BLEU actually measures, or rather how it *compares*. A comparison between a translation and a reference is done by breaking down sentences into so-called n-grams, where n denotes the number of words in the phrase which is examined. The n-grams from the new translation are compared with the n-grams of the reference. A perfect match between a translation and a reference would yield a score of 1, while a lack of any similarity would be scored 0. This measure of similarity is usually rounded and multiplied by 100 to make it more readable as a score between 0 and 100. For example a score of 0.25421 would be expressed as a BLEU score of 25.4. This particular score would mean that roughly one quarter of the n-grams in the new translation corresponds with the reference. Since we measure the similarity to a reference, or several references, the absolute score has no real meaning in terms of translation quality from a linguistic point of view. The score of one translation will change if the reference is changed or if more references are used.

Anyone, who is proficient in a second language, is aware of the fact that many, especially complex phrases, can be translated in different ways, so a relation of 1:n between source and target language is not uncommon in terms of acceptable translations. Needless to say, a metric based solely on the comparison of patterns cannot take this into account. Therefore, one has to be very careful about using the term "quality" when talking about BLEU scores, since we compare to a reference translation, considering only structure and lexical properties and not how well the meaning was carried over into its new representation. A translation differing greatly from a reference in word choice and word order could be an equally good alternative way to carry over that meaning; it might be even better, but will be rated poorly by the automatic evaluation method. Hence a good translation might potentially receive a very low score, if it happens not to correspond with the reference (Zhang et al., 2004).

Keeping this in mind, it is rather obvious that the popularity of BLEU is not based on its accurate classification of good translations, but the ease of use and the quick results even for very large volumes of data. One might argue that the way translations are scored by BLEU implicitly favors automated translation, especially SMT and NMT, since these methods rely on statistical analysis of corpora, hence produce results which BLEU "expects to see". One could also argue that BLEU evaluation favors standardization of translation by assigning higher scores to results which adhere to certain patterns. This potentially leaves out many good options of translation but can be advantageous for specific applications, for example the aforementioned user manuals.

Further, these automated methods are the only way for a quick first assessment of new MT approaches, and are very efficient in helping to tweak parameters while developing MT systems. Speed of assessment, convenience, and financial feasibility make these methods a necessity in rapid MT development and fine-tuning. However, one has to keep in mind the serious shortcomings, mentioned in the previous paragraphs. A deeper and more detailed discussion of these limitations can be found in (Zhang et al., 2004).

Most recently word and sentence embeddings with *BERT* (Devlin et al., 2018) have been extensively used for cross-lingual information retrieval (Jiang et al., 2020), and certainly can be utilized to address the issue of finding similarities between sentences in two languages. These methods show very good results and there is promising research on models which work for any language pair (Feng et al., 2020). However, the same requirements apply for these models as mentioned in Section 2.3 for NMT: Large volumes of training data are needed as well as long training times. Additionally, these models are black box, so traceability is a problem.

The approach taken by (Utiyama and Isahara, 2003) in order to create one of the highest quality Japanese-English parallel corpora, the *JENAAD* corpus, listed in Table 2.1, was independent of example data and relied on one of the most successful text-retrieval algorithms *BM25*, derived from the Probabilistic Relevance Framework (PRF), dating back to research work in the 1970-1980s, described in (Robertson and Zaragoza, 2009).

To summarize, all these above mentioned methods have their advantages and disadvantages. In any case, it is important to remember that *machine* evaluation of *machine* translation must not be the final judgement of quality. We should be aware that haphazard assessments of training data will result in a propagation of low text quality, since future resources will partially be relying on previously MT processed data. As stated in the beginning of this section, only human expert assessment can be completely relied upon, for final judgement.

Simplicity is the soul of effciency.

Austin Freeman

3 AWCAT Framework

3.1 Overview

As stated in Section 1.1.1, one of the goals of this thesis is to empirically examine whether and how much of Wikipedia content can be used as a source of parallel corpora for a specific language pair. By developing the framework

Automated Wikipedia Corpus Acquisition Tool (AWCAT) we created a data collection allowing for an insight into this issue for English-Japanese.

We processed this data collection into a parallel corpus, and measured the yield. This process can be repeated and further comparisons can be made for different domains by giving the software framework different initial seed topics. Most importantly, we addressed these research challenges with a transparent and traceable approach, offering a counterbalance and potential enhancement of state-of-the-art methods that often solely rely on a black box neural network approach, and require large volumes of training data.

We present the software framework in this section with a broad overview and name all language resources (Section 3.2) and tools (Section 3.3) which we used to create the framework. In Chapters 4, 5, and 6 we describe each stage and its modules in detail by listing important segments of the source code and explain the thoughts and theory that motivated our approach. In Section 3.4 we list the specifications of the hardware used for coding and running the software.

Functions of AWCAT are packaged in modules, which themselves belong to stages in the overall architecture. This modular approach allows each stage and every module to be functional by itself and offers good readability and flexibility following a good software engineering practice. The good readability makes the code easier to maintain, and the independent stages and modules allow for easier change and adjustment to other requirements such as different language pairs.

We segmented the software modules into three stages. The Data Extraction Stage, the Data Preparation Stage, and the Sentence Alignment Stage. An overview of the stages and the modules assigned to them is shown in Figure 3.1.

In the Data Extraction Stage we collect the raw data from the assigned Wikipedia pages. Rather than accessing data at random, for the performance reasons mentioned in Section 1.1.1, a seed of topic is taken as input and the crawling process follows a topic link-based algorithm described in Section 4.2.

Once the data is collected, we pass it to the Data Preparation Stage where it is formatted and prepared for alignment. This step allows to dispose of unnecessary data to reduce CPU hours in the later steps.

During the Sentence Alignment Stage we compare the Japanese sentences to the English sentences according to several criteria described in Chapter 6. The similarity is quantified in a metric explained in Section 6.2.

The separation of modules follows a division of self-contained tasks to of-

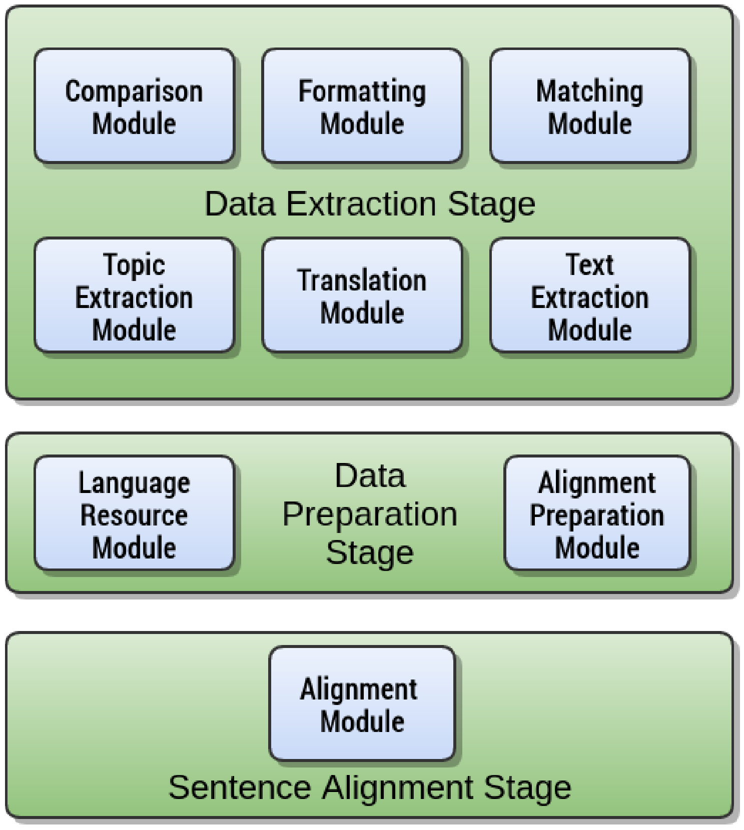


Figure 3.1: Stages with their corresponding modules.

fer flexibility for adjustments such as parameter tuning, adjustments for specific domains and other language pairs. In Chapters 4, 5, and 6 we describe the modules of this architecture in incremental levels of detail. We start with an overview of the stage, followed by the modules, the functions within these modules, and finally details of key parts of the source code.

3.2 Language Resources

The lexical resources used for Japanese are *edict2*, and *JMnedict*. *edict2* is a collection of over 180,000 words (at the time of writing) and common multi-word expressions with their English translations. We considered using *JMdict* being an XML version of *edict2*. The reason why we chose the latter is to avoid the XML overhead during the conversion. *JMnedict* is a Japanese multilingual named entity dictionary file. URLs to these resources are listed in Table 3.1.

In order to achieve a performance boost, by avoiding multiple lookups, we use the JSON file format to build dictionaries, which we then reuse throughout the execution of the program and for subsequent program executions with different parameters.

Name	Version	URL
edict2	current as of October 2019	`http://edrdg/jmdict/edict.html`
JMnedict	current as of October 2019	`https://www.edrdg.org/enamdict/enamdict_doc.html`

Table 3.1: Language resources used in our framework.

3.3 Software, Programs, Tools

The software is written in the *Python* programming language Version 2.7 and takes advantage of libraries such as the *Natural Language Toolkit (NLTK)*, the *Wikipedia* library, the HTML/XML parsing library *BeautifulSoup*, and the regular expression library *re* for efficient and flexible character comparisons. Python's extensive and straightforward *codecs* library enables the encoding and decoding of *Kanji* (Japanese characters) as well as other characters represented in Unicode. The standard *NLTK* libraries are used to tokenize, lemmatize, and PoS-tag the English corpus.

The resources used for Japanese are *MeCab*, which is an open-source PoS-tagger for Japanese. The output of *MeCab* is used to identify signal words, foreign words, names and numbers. The version numbers and URLs are listed in Table 3.2.

Name	Version	URL
NLTK	3.5	https://nltk.org
MeCab	0.996	https://taku910.github.io/mecab

Table 3.2: Language tools used in our framework.

3.4 Hardware

We implemented the framework on a desktop computer. Especially during the alignment process, which took the most computing time, we executed the code on both a desktop computer, and a laptop computer. We also were granted access to the 3rd version of the *Vienna Scientific Cluster* (VSC-3), which we extensively used for much of the framework runtime. The hardware specification of the two computers are shown in Table 3.3, the information about the VSC-3 can be found online at the *Vienna Scientific Cluster* website[*].

Harddrive reading speeds are obtained with the linux command shown in Code 3.1.

```
# hdparm -tT /dev/sda
```

Code 3.1: Linux Command Line: HDD reading speed test.

[*] https://vcs.ac.at/systems/vcs-3/ (Last accessed in August 2020.)

	Desktop Computer	Laptop Computer
CPU and frequency	Intel Core i5 @ 3.6Ghz	Intel Core i7-5500U @ 2.4GHz
RAM and frequency	8GB @1333 MHz	8GB @1600MHz
HDD reading speed	126 MB/sec	1112 MB/sec

Table 3.3: Specifications of hardware used for experiments.

Getting information off the Internet is like taking a drink from a fire hydrant.

Mitchell Kapor

4 Data Extraction

4.1 Data Source – Wikipedia

One of the biggest challenges for collecting data is the selection of the source. Ideally, the source should offer enough data, the desired quality, and a consistent structure, which allows for easy access.

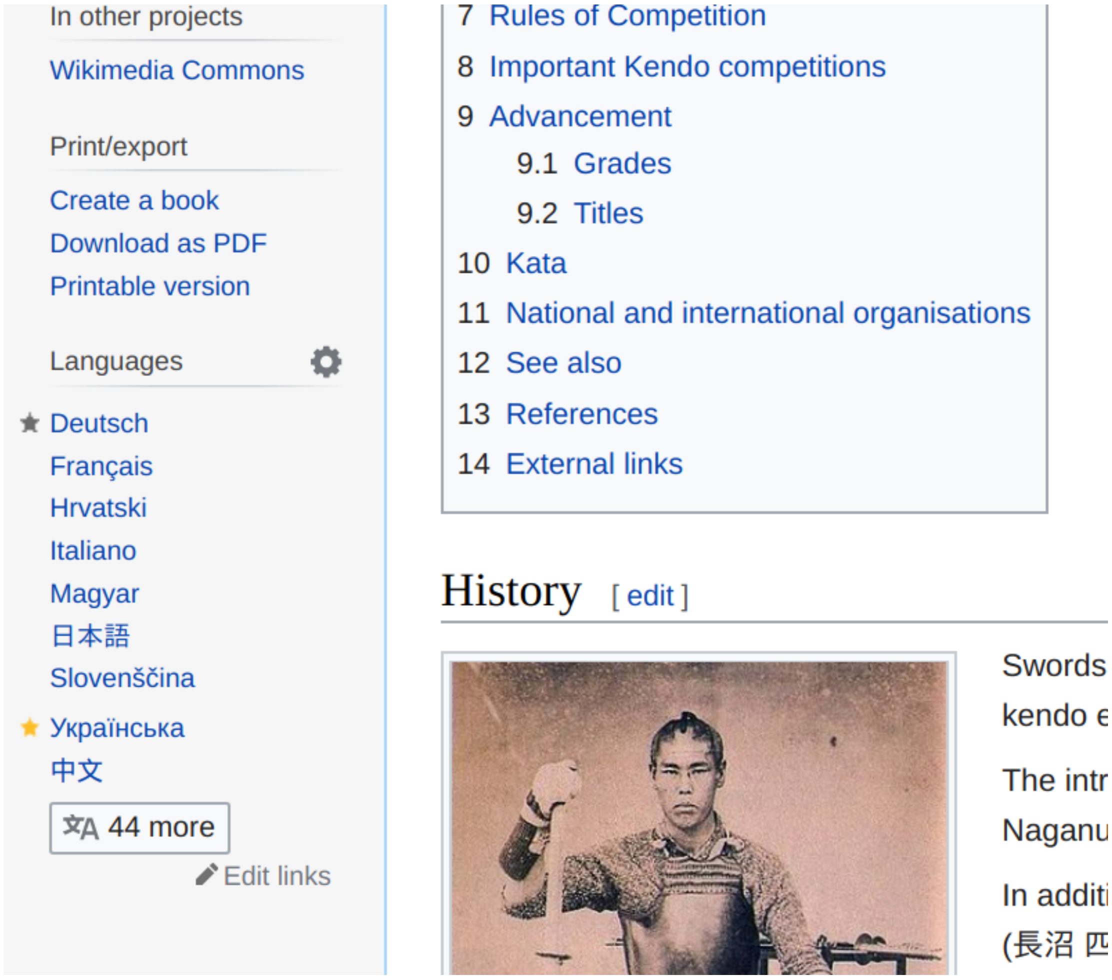
While Wikipedia might not be the best source for the highest quality and accurate translations, it offers large volumes of data and is well organized, due to its semantic HTML structure.

The Wikipedia pages in two languages for one article are semantically linked. This eliminates the need for additional dictionary lookups, it even functions as a dictionary, since terms are automatically disambiguated.

An example of the language link selection for an article is shown in Figure 4.1. On the left side of the page, or in a special drop-down menu for the mobile version of Wikipedia, a list is shown with all languages in which this article is available. Clicking on one of these language links takes the user to the article in that language. The structure behind this functionality is coded in JSON files, which contain link addresses to all articles in other languages. An excerpt from such a file is shown in Figure 4.2.

The source language for the entry "Kendo" is English, and all available language links are listed. The list in the figure is abbreviated and the Japanese entry is added at the bottom. It is indicated by the "lang:ja" field, whereas the hexadecimal numbers "\u5263\u9053" are the unicode encoding, a so-called *code point*, for the Japanese word "剣道" which stands for Kendo.

The semantic layout of pages is consistent, hence identifying titles, figures,


In other projects

Wikimedia Commons

Print/export

Create a book
Download as PDF
Printable version

Languages

★ Deutsch
Français
Hrvatski
Italiano
Magyar
日本語
Slovenščina
★ Українська
中文

文A 44 more

Edit links

7 Rules of Competition
8 Important Kendo competitions
9 Advancement
9.1 Grades
9.2 Titles
10 Kata
11 National and international organisations
12 See also
13 References
14 External links

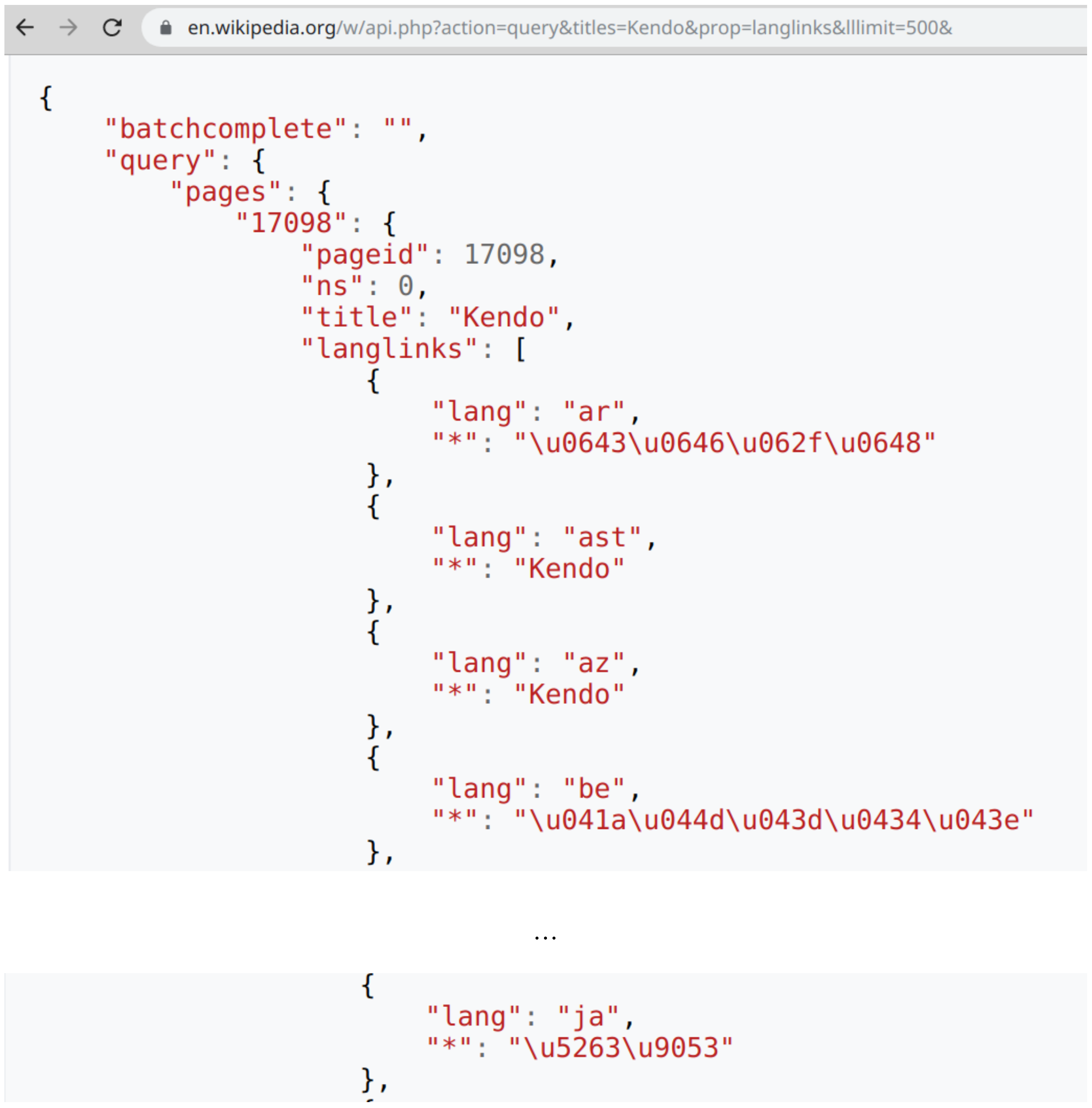
History [edit]

Swords
kendo e

The intr
Naganu

In additi
(長沼 四

Figure 4.1: Wikipedia page language links.


en.wikipedia.org/w/api.php?action=query&titles=Kendo&prop=langlinks&lllimit=500&

```
{
    "batchcomplete": "",
    "query": {
        "pages": {
            "17098": {
                "pageid": 17098,
                "ns": 0,
                "title": "Kendo",
                "langlinks": [
                    {
                        "lang": "ar",
                        "*": "\u0643\u0646\u062f\u0648"
                    },
                    {
                        "lang": "ast",
                        "*": "Kendo"
                    },
                    {
                        "lang": "az",
                        "*": "Kendo"
                    },
                    {
                        "lang": "be",
                        "*": "\u041a\u044d\u043d\u0434\u043e"
                    },

                                    ...

                    {
                        "lang": "ja",
                        "*": "\u5263\u9053"
                    },
```

Figure 4.2: Wikipedia page language links JSON file.

menus, etc. can be done very easily with existing libraries. In this thesis we took advantage of the Python library *BeautifulSoup*.

While structure, semantic layout and abundant data are a clear advantage, the downside of Wikipedia for parallel corpora extraction is that an unknown portion of the content is not translated, but rather created independently across languages. Only a portion is translated by professionals and even those are not always good, accurate translations.

Occasionally, translation mistakes or stylistic errors spanning over several pages can be observed. They are sometimes very subtle and often go unnoticed. A notable example of this is the use of a certain tense when describing historical figures in English and French. In English, one uses past tense, whereas in French present tense is correct. Whenever such content is translated from English to French, which happens more often than the other way around, the past tense tends to find its way into the French version.

Such seemingly small but significant mistakes are common on Wikipedia across many language pairs and have to be taken into consideration.

Additionally, the content of Wikipedia can be highly asymmetrical across languages. Depending on the popularity of a topic in the given country, region, or culture, an article for a certain topic can be very extensive in one language, while being scarcely represented in another. Further, due to the dynamic nature of Wikipedia, it is difficult to consistently examine these symmetries or the lack thereof.

4.2 Selective Crawling

Gathering data in such a way that we send as little useless data to the next step as possible is a big challenge. We have addressed this challenge with selective crawling, described in this section.

Additionally to the preselection in terms of parallel data candidates, this method allows us to extract data within a certain domain, given a list of *seed topics*.

We solved the task of efficiently identifying text, which is likely to contain similar, same, or translated content by determining the ratio of article links shared between the articles in Japanese and English. The bigger the ratio of links that point to the same articles from both the Japanese and the English article version, the more likely the content is to be paraphrased or translated.

For example, let J_a be a Japanese article of topic a, and E_a its English article equivalent. If J_a contains five links to J_b and eight links to J_c, and E_a contains four links to E_b and seven links to E_c, the ratio would be $r = 11/13 = 0.846$.

We defined a threshold for the ratio, which is a cutoff value for the pre-selection. This threshold value can be increased or decreased for further empirical studies. A higher value results in a higher chance of a translated or paraphrased article, a lower value yields more candidate data. The final step is the selection of articles which score above this ratio of shared links, and the extraction of their article text with the Python library *Wikipedia*.

In the process of extracting and pre-selecting articles, we noticed that the lists of article titles in Japanese and English can very easily be converted into a bilingual glossary of terms for a certain domain. Such a quick source of translated terms is quite handy for many applications in translation science and industry and other language technology applications. Such a list can be obtained in minutes by running the first part of the framework with a topic seed of choice. We describe this coincidental but handy byproduct in Section 4.4.

4.3 Implementation of the Data Extraction Stage

The Data Extraction Stage consists of six modules, which are explained in detail in this section. A schematic of the sequence of processing is shown in Figure 4.3.

Each module in this schematic is highlighted with a bold face font and the

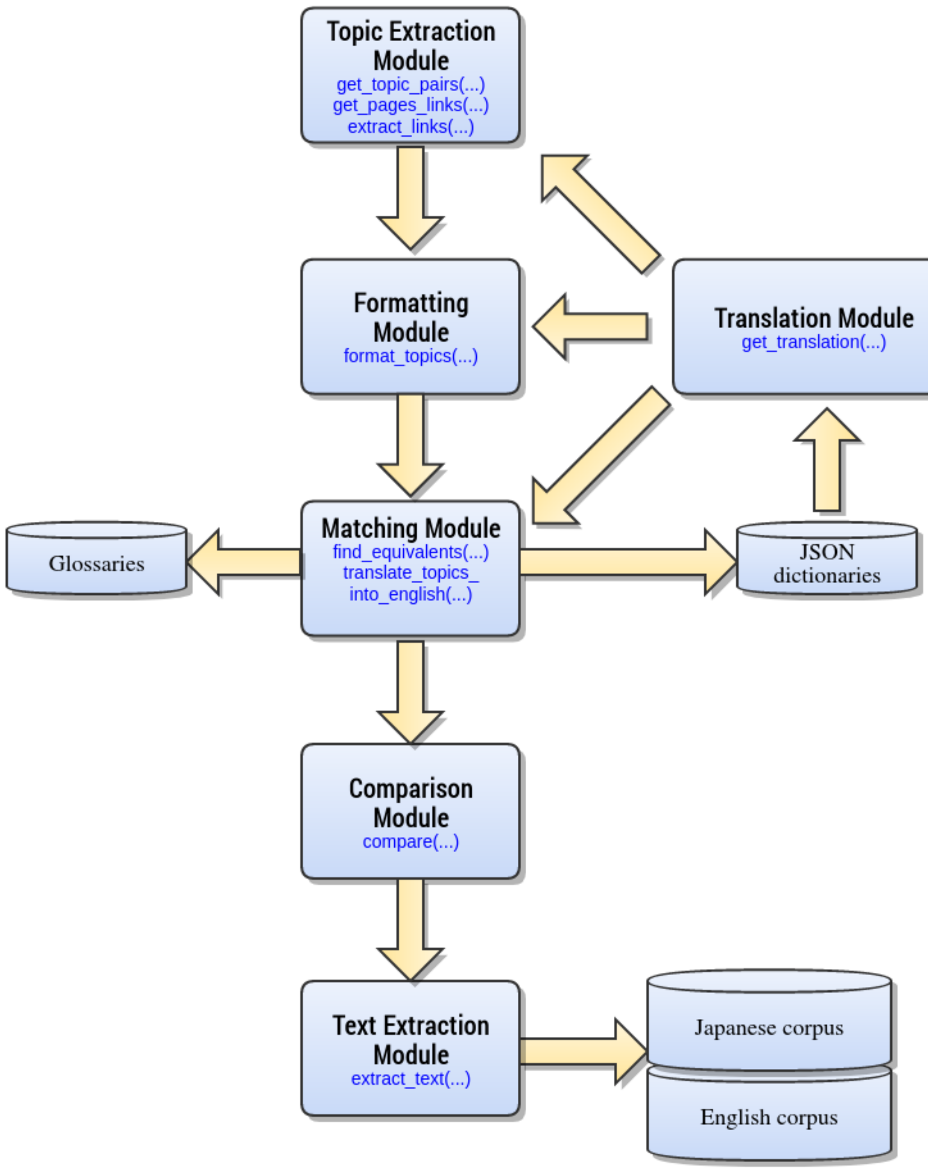


Figure 4.3: Modules of the Data Extraction Stage for selective harvesting and text extraction from Wikipedia articles. The goal of this chain of modules is to obtain candidate sentences for a parallel corpus. Module names are written in bold face, function names (without parameters) are shown in small, blue font.

functions listed below in blue. The cylindrical containers in the graphic symbolize text resources. At the beginning of the stage, dictionary resources are used as input. The Matching Module produces a glossary and a JSON dictionary. This dictionary is growing during each iteration of the framework and allows for time savings by avoiding online lookups. Additionally, it is utilized for efficient translation lookups in the Sentence Alignment Stage. The output of the Data Extraction Stage is text data in the form of a Japanese corpus and an English corpus. These two data collections are the candidates for a parallel corpus.

In the following subsections, we include important pieces of source code, libraries, and external programs; and describe their role and significance in the framework. The displayed lines of source code are denoted with "Code" followed by a sequential number within the chapter; they include line numbers for detailed referencing. For the sake of brevity, the line references are attached to the code number with a colon; for example, Code 5.2:3 would be line 3 of the second Code snippet in Chapter 5. It is important to mention that these code snippets are not always shown as complete functions, but rather as parts of functions. Whenever a small part of a function is presented, or it is abbreviated by cutting out lines for the purpose of brevity and readability, this cut is identified by "...". Whenever this is done, the line numbers are continued in sequence as the code snippet is presented but not in the sequence in which they are written in the implementation of the framework. The complete source code can be found in the Appendix.

4.3.1 Topic Extraction Module

The first module is the Topic Extraction Module. The libraries imported at this point are shown in Code 4.1. These libraries are also used for all following modules, throughout the entire stage.

```
from bs4 import BeautifulSoup
import requests
import codecs
import re
import json
import urllib
import os
```

Code 4.1: Preamble: Libraries for Data Extraction Stage.

BeautifulSoup is a library for extraction of XML data. Traversal of the Wikipedia page structure is very straightforward with this library. We use the *requests* library to fetch the HTML pages from which data will be extracted. The *codecs* library is important throughout the entire framework, since it is vital for the conversion of Japanese characters into a utf-8 representation. The same is true for the regular expression library *re* for efficient and flexible character comparisons. We use the *json* library to read the files containing the links to the equivalent articles in the other language. The *urllib* library provides wrappers for URL encoding, and the *os* library for directory changes in the framework directory structure to sort output files at run-time, to avoid cluttered directory structure.

The first step in the process of extraction is the seeding with initial topics. The function **`get_topic_pairs`** in the Topic Extraction Module takes a list of strings, which should be valid English Wikipedia entries, see Code 4.2:1 (line 1 of Code 2), cleans the strings of unnecessary white spaces and passes them to the function `get_translation` of the Translation Module, where the equivalent Japanese articles are retrieved (Code 4.2:6). The Translation Module is also accessed from several other modules and is described in detail in Section 4.3.2.

```
1 def get_topic_pairs(topic_list):
2     # for each start topic
3     for topic in topic_code:
4         start_topic= topic.strip()
5         #get start topic in japanese by ID check
6         topic_ja = get_translation(start_topic,'ja','en')
```

Code 4.2: Function `get_topic_pairs`: Extracting topics which will be later crawled for data.

The next step prepares the output files for the lists of topics, which are selected for data collection (Code 4.3). Explicit utf-8 encoding has to be performed during reading and writing of Japanese characters in Python. It is also important to note that each Python file that will process utf-8 data has to include a special line in the preamble, usually right after the definition of the Python environment; this is shown in Code 4.4. It should be mentioned that this is no longer necessary in Python 3, however, we still use Python 2.7 for our implementation.

```
        #open files to store subtopics
        ftopics_en = codecs.open('data/'+start_topic+'_topics_en.\
txt','w', encoding='utf8')
        ftopics_ja = codecs.open('data/'+start_topic+'_topics_ja.\
txt','w', encoding='utf8')
```

Code 4.3: Function `get_topic_pairs` (cont.): Using codecs to write Japanese characters.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

Code 4.4: Preamble: Defining the Python environment and the utf-8 encoding.

The next section of this function, shown in Code 4.5 extracts the topic with a call to the function `get_pages_links`, which takes the initial topic and the language parameter (‘en’ for English, ‘ja’ for Japanese) as arguments. The data returned by this function is then written to the previously opened text files.

```
        #call get_pages function to get subtopics for english
        topics = get_pages_links(start_topic, 'en')
        #write results to file and close output file
        for topic in topics:
            ftopics_en.write(topic[0]+'->'+topic[1]+'\n')
        ftopics_en.close()
        #call get_pages function to get subtopics for japanese
        topics = get_pages_links(topic_ja, 'ja')
        #write results to file and close output file
        for topic in topics:
            ftopics_ja.write(topic[0]+'->'+topic[1]+'\n')
        ftopics_ja.close()
```

Code 4.5: Function `get_topic_pairs` (cont.): English and Japanese articles are written to text files.

We will take a closer look at the **`get_pages_links`** function, since here we select the pages, which are linked from the initial seed topics. They will be selected for data extraction, determining how much data will be sent to the next stage. The construction of the URL pointing to the page with the article that is to be extracted is seen in Code 4.6.

```
# return all links on the topic page and all subsequent links
def get_pages_links(topic, lang):
    start_url = 'https://'+lang+'.wikipedia.org/wiki/'
    domain = 'https://'+lang+'.wikipedia.org'
    start_url=start_url+topic #main topic link
```

Code 4.6: Function `get_pages_links`: Construction of an article URL by assigning language and topic.

```
# get own title, link titles and links for main topic
title, ext_titles, ext_links = extract_links(url=start_url)
# store in items list
items.extend(zip([title]*len(ext_titles), ext_titles))
for ext_link in ext_links:
    # omitting Wiktionary entries and pronunciation links
    if 'wikt' not in ext_link and 'Help:IPA' not in ext_link:
        try:
            # resolve encoding issues
            ext_link=urllib.unquote(ext_link).decode('utf-8')
            # get own title, link titles
            # and links for main topic
            title, ext_titles, ext_links = extract_links(domain + \
   ext_link)
            # store in items list
            items.extend(zip([title]*len(ext_titles),ext_titles))
        except UnicodeEncodeError, e:
            print('UnicodeEncodeError at: ',ext_link,'-reason:', \
   str(e))
            pass
    if len(items) > 2500:
        break
return items
```

Code 4.7: Function `get_pages_links` (cont.): Extracting links and their titles.

In Code 4.7:2 the call to the function `extract_links` returns a triple of values which are the title of the current page, a list of titles of links on the current page, and the URLs of the links on the current page.

The next line stores these tuples in a list. For every link we have extracted on that page (Listing4.7:5), we repeat the link extraction, so that with every iteration we grow the list by the set of the links from the next page. We do not want to include Wiktionary entries and pronunciation links, since they are highly language dependent and would introduce errors in the ratio of comparable links.

The `try,except` block at Code 4.7:8,18 is crucial, since unexpected variations in spelling of the article name can cause this part of the program to crash. Additionally, since URLs are restricted to a set of characters belonging to the US-ASCII set, passing a Japanese character as a URL string would cause an exception and also crash the program. We use *urllib* to convert Japanese titles of articles into a URL representation of the Japanese characters, which is a hexadecimal representation. In Table 4.1 we show an example of a URL pointing to the article “Airplane” in English Wikipedia, the URL pointing to the Japanese page for the same topic, and the representation of this address in US-ASCII. The `try,except` block is set up so that the program will skip the iteration if there is an error, such that the program can be safely run in the background without the need for constant monitoring.

URL to English article Airplane
https://en.wikipedia.org/wiki/Airplane
URL to Japanese article Airplane (飛行機 =hikōki)
https://ja.wikipedia.org/wiki/飛行機
URL to Japanese article Airplane in US-ASCII ecoding
https://ja.wikipedia.org/wiki/%E9%A3%9B%E8%A1%8C%E6%A9%9F

Table 4.1: Example of URL US-ASCII encoding.

While most browsers will take care of this conversion, a direct call with the Python *requests* library does not, therefore, it needs to be done manually.

The **`extract_links`** function is shown in Code 4.8. This function does the actual fetching of links from the HTML document. The request call gets the previously constructed URL and converts the content from an XML format to a *BeautifulSoup* object (Code 4.8:13–16). The next line filters the content by paragraphs, followed by a filter for invalid links, and a filter for the HTML tag `href`, which denotes external links. The tag `title` is extracted next, which is the name of the link (in our case the article topic) (Code 4.8:13–

```
 1 # return a list of links to other Wikipedia articles
 2 def extract_links(url):
 3 # get soup with lxml parser
 4     soup = BeautifulSoup(requests.get(url).content,'lxml')
 5     p_tags = soup.findAll('p')# find all paragraph tags
 6     # gather all <a> tags
 7     a_tags = []
 8     for p_tag in p_tags:
 9         a_tags.extend(p_tag.findAll('a'))
10     # filter the list : remove invalid links
11     a_tags = [a_tag for a_tag in a_tags if 'title' in a_tag.attrs \
   and 'href' in a_tag.attrs and not 'class' in a_tag.attrs]
12     # get article titles
13     titles = [a_tag.get('title') for a_tag in a_tags]
14     # get article links
15     links  = [a_tag.get('href') for a_tag in a_tags]
16     # get own title
17     self_title = soup.find('h1', {'class' : 'firstHeading'}).text
18 return self_title, titles, links
```

Code 4.8: Function `extract_links`: Extracting links and filtering the content

16), and in the last line of the function the title of the current page is obtained (in our case the current article).

The triple, returned from Code 4.8:19 to Code 4.7:13 is extracted until a certain number of pages and their links is reached (Code 4.7:22), we chose 2500 for this particular experiment. This number is arbitrary and can be chosen depending on the available CPU-time.

4.3.2 Translation Module

The **`get_translation`** function in this module takes an article title, a source language code, and a target language code as function parameters. The language codes are the standard Wikipedia abbreviations, i.e. 'ja' for Japanese and 'en' for English. In Code 4.9:4 we build a URL with the input parameters and pass this string to a request. With the returned data we create a JSON object. We traverse the hierarchical JSON structure until we reach the "langlinks" level, see Figure 4.2, and find the appropriate entry, according to the input parameter. This entry is the article title. The `try/error` blocks are important to deal with inconsistent JSON files, which tend to occur occasionally.

4.3.3 Formatting Module

The **`format_topics`** function makes up the Formatting Module, in which we reformat the output of the previous module. The reason why it is separated as a distinct module is that it acts as a buffer in case a different format is needed after the extraction. It serves in essence as a format interface. Additionally to that, it is also a preparation to the next module. In the process we translate the Japanese article titles and the titles of the linked articles to English

```python
# return the Wikipedia site equivalent in a target language
def get_translation(topic,source_lang,target_lang):
    # use Wikipedia's json database to look it up
    json_url='https://'+source_lang+'.wikipedia.org/w/api.php?\
action=query&titles='+topic+'&prop=langlinks&lllimit=500&format\
=json'
    content = requests.get(json_url).content
    json_data = json.loads(content)
    item=''
    # iterate through json hierarchy to find langlinks category
    try:
        for i in json_data["query"]["pages"]:
            pageid=i
    except KeyError, e:
        print('KeyError at topic:',topic,' - reason: ',str(e))
        pass
    except TypeError, e:
        pass
    try:
        for i in json_data["query"]["pages"][pageid]["langlinks"]:
            # in langlinks category, find desired language
            if i['lang']==target_lang:
                # there is the topic equivalent
                item = i['*']
    except KeyError, e:
        print('Keyerror at topic',topic,' - reason: ',str(e))
        pass
    except TypeError, e:
        pass
return item
```

Code 4.9: Function `get_translation`: Finding article titles in another language.

by passing them to the Translation Module.

4.3.4 Matching Module

In this module we identify the articles which have an equivalent in both English and Japanese with the **`find_equivalents`** function. The names of the text files in the data directory are the titles of the articles. Each file contains a list of articles that are linked from its text. At this point, the Japanese articles are represented in English. We point out again that these are not translations, but article name equivalencies obtained from Wikipedia in Code 4.9.

This simplified structure of the collected titles allows us to quickly identify the articles which are available in Japanese and English. This check is shown in Code 4.10. We find the matching articles in Code 4.10:12, and store them in a separate directory for further processing.

At this point, we have a collection of English and Japanese representations of Wikipedia articles. We take advantage of this collection to build a glossary of translations by the function `translate_topics_into_english` (Code 4.11), which will be very handy later. Since all the terms in this collection relate to the seed topics, they are likely to appear often in the extracted texts and hence during the alignment process. The glossary created at this point

```
def find_equivalents():
    file_list_ja=[]
    file_list_en=[]
    os.chdir('./data/topics') # change to data dir
    for file in glob.glob('*_ja.txt'): # for every japanese file
        file_list_ja.append(file[:-7]) # get topic from filename
    for file in glob.glob('*_en.txt'): # for every english file
        file_list_en.append(file[:-7]) # get topic from filename
        #store data in pairs
    for item in file_list_en:
        if item in file_list_ja:
            copy('./'+item+'_ja.txt','./pairs/'+item+'_ja.txt')
            copy('./'+item+'_en.txt','./pairs/'+item+'_en.txt')
            topic_pairs.append(item)
    os.chdir('../../') # back to main dir
```

Code 4.10: Function `find_equivalents`: Storing equivalent articles.

not only serves as a quick access dictionary, but eliminates potential problems with polysemy while using a regular dictionary. At the same time this glossary is used in repeated executions of this stage. The locally stored JSON file is a very quick alternative to comparatively long lookups in the online Wikipedia JSON language directory described in Section 4.1, and depicted in Figure 4.2.

4.3.5 Comparison Module

In this module, we determine with the **`compare`** function which articles are to be defined as parallel data candidates, as described in Section 4.2.

```python
def translate_topics_into_english():
    file_list=[]
    os.chdir('.')
    # if there is no dictionary file, open a new one
    if not os.path.exists('./topics_dict_ja_en.json'):
        empty_dict={}
        f=open('topics_dict_ja_en.json','w')
        json.dump(empty_dict,f)
        f.close()
    topic_dict_ja_en={}
    with codecs.open('topics_dict_ja_en.json','r',encoding='utf8')\
     as fdict:
        topic_dict_ja_en=json.load(fdict)
    # translate sorted Japanese files
    file_list=[]
    for file in glob.glob('data/topics/pairs/*_ja.txt'):
        file_list.append(file)
    for file in file_list:
        fout=codecs.open(file[:-4]+'_en_ja.txt','w','utf-8')
        with codecs.open(file,'r','utf-8') as f:
            lines=f.readlines()
            for line in lines:
                try:
                    fout.write(topic_dict_ja_en[line[:-1]]+'\n')
                except KeyError, e:
                    print('KeyError: ', str(e))
                    trans=get_translation(line[:-1],'ja','en')
                    fout.write(trans+'\n')
                    topic_dict_ja_en[line.rstrip()]=trans
                    print(line.rstrip()+'->'+trans+' added to \
    dictionary')
                    pass
    f=open('topics_dict_ja_en.json','w')
    json.dump(topic_dict_ja_en,f)
    f.close()
```

Code 4.11: Function `translate_topics_into_english`: Storing topics translations in a JSON file.

The corresponding files containing the article links in English and Japanese are compared and the number of matching links is counted (Code 4.12:17). We compute the ratio in (Code 4.12:20) with respect to the total number of articles. If this ratio meets the threshold (Code 4.12:25), we add the article to a list, which is returned at the end of this function and becomes the input for the final module in this stage, the Text Extraction Module.

4.3.6 Text Extraction Module

In this module, we traverse the list of articles that have equivalents above a certain threshold value and extract their text with the **`extract_text`** function (Code 4.13). Thanks to the convenient *Wikipedia* library, this process is straightforward, and merely requires a `try/else` block to make sure the program does not stop at an error caused by an inconsistency in a Wikipedia article, or a transmission error. The result of this module are two collections of texts, one in English, the other in Japanese, which contain the parallel candidates.

```
    #open both files and compare
    for item in file_list_en:
        common_counter=0
        topics_ja=[]
        topics_en=[]
        with codecs.open(item+'_ja_en_ja.txt','r','utf-8') as fja:
            lines=fja.readlines()
            for line in lines:
                topics_ja.append(line.split())
        with codecs.open(item+'_en.txt','r','utf-8') as fen:
            lines=fen.readlines()
            for line in lines:
                topics_en.append(line.split())
        for topic_ja in topics_ja:
            for topic_en in topics_en:
                if topic_ja==topic_en:
                    common_counter+=1
                    break
        # calculating ratio, after counting extracted links for \
each topic
        if len(topics_ja)>0:
            scounter+=1
            similar.append(item)
            ratio = float(float(common_counter)/float(len(\
topics_ja)))
            if ratio>0.7: # ratio threshold value
                print str(common_counter)+' link matches in topic \
>'+ item + '< out of total '+ str(len(topics_en))+' links -> \
ratio: '+ str(round(ratio,3))
    print 'Total similar pages count: '+str(scounter)
    return similar
    os.chdir('../../../') # back to main dir
```

Code 4.12: Function `compare`: Comparing articles.

```
def extract_text(link_list):
    os.chdir('./data/topics/pairs')
    ftext=codecs.open('text_english.txt','w','utf-8')
    for item in link_list:
        try:
            # passing article name to get reference to page
            p = wikipedia.page(item.strip())
            ftext.write(p.content) # getting text from Wikipedia \
   page
        except wikipedia.exceptions.WikipediaException as e:
            pass
        scounter-=1
    ftext.close()

    # same for Japanese
    ftext=codecs.open('text_japanese.txt','w','utf-8')
    for item in link_list:
        wikipedia.set_lang('ja')
        try:
            # passing article name to get reference to page
            p = wikipedia.page(get_translation(item.strip(),'en','\
   ja'))
            ftext.write(p.content)
        except wikipedia.exceptions.WikipediaException as e:
            pass
        scounter2-=1
    ftext.close()
    os.chdir('../../../') # back to main dir
```

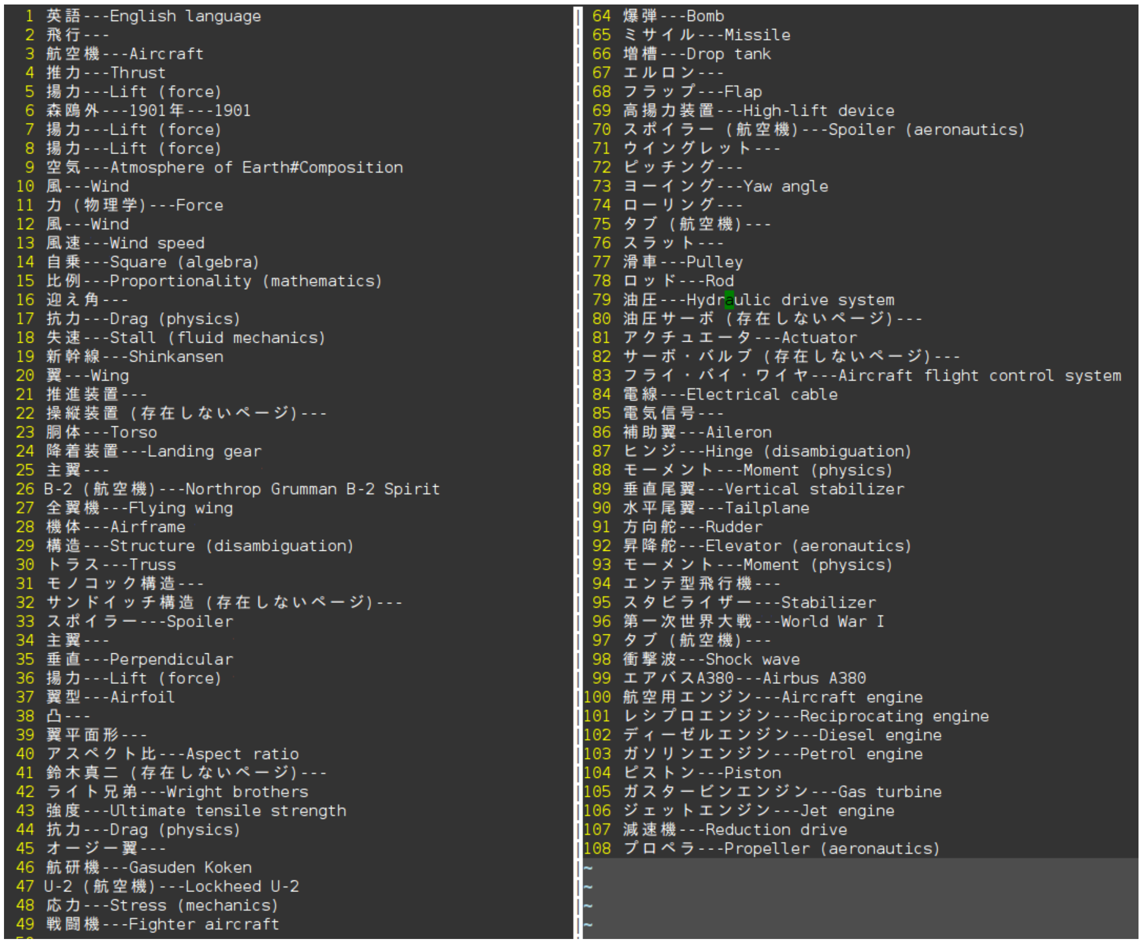
Code 4.13: Function `extract_text`: Scraping text of selected pages.

4.4 Building Glossaries

The lists of article topics in both languages derived from an initial seed of topics and selected based on common link similarity (described in chapter 4.2) results in topic related lists of words. This by-product of topic selection is a quick way to build glossaries according to the initial seeds. These glossaries of English-Japanese article topics offer the advantage of being semantic equivalents as defined by Wikipedia in conjunction with a certain domain (as defined by the initial seed), rather than translations from a dictionary, which often require to choose between several possible translations.

One possible application of such a collection, which can be created dynamically and on-demand with any desired seed of topics, is a glossary preparation for translators and interpreters. This is especially true for interpreters who prepare for assignments in a specialized domain, e.g. a talk on a specific technology, a medical topic, or a political debate, such a dynamic bilingual glossary for a certain topic domain can potentially improve their preparation.

An example of such a glossary built by the seed topic "Airplane" is shown in Figure 4.4.



```
英語---English language
飛行---
航空機---Aircraft
推力---Thrust
揚力---Lift (force)
森鴎外---1901年---1901
揚力---Lift (force)
揚力---Lift (force)
空気---Atmosphere of Earth#Composition
風---Wind
力 (物理学)---Force
風---Wind
風速---Wind speed
自乗---Square (algebra)
比例---Proportionality (mathematics)
迎え角---
抗力---Drag (physics)
失速---Stall (fluid mechanics)
新幹線---Shinkansen
翼---Wing
推進装置---
操縦装置 (存在しないページ)---
胴体---Torso
降着装置---Landing gear
主翼---
B-2 (航空機)---Northrop Grumman B-2 Spirit
全翼機---Flying wing
機体---Airframe
構造---Structure (disambiguation)
トラス---Truss
モノコック構造---
サンドイッチ構造 (存在しないページ)---
スポイラー---Spoiler
主翼---
垂直---Perpendicular
揚力---Lift (force)
翼型---Airfoil
凸---
翼平面形---
アスペクト比---Aspect ratio
鈴木真二 (存在しないページ)---
ライト兄弟---Wright brothers
強度---Ultimate tensile strength
抗力---Drag (physics)
オージー翼---
航研機---Gasuden Koken
U-2 (航空機)---Lockheed U-2
応力---Stress (mechanics)
戦闘機---Fighter aircraft
```

```
爆弾---Bomb
ミサイル---Missile
増槽---Drop tank
エルロン---
フラップ---Flap
高揚力装置---High-lift device
スポイラー (航空機)---Spoiler (aeronautics)
ウイングレット---
ピッチング---
ヨーイング---Yaw angle
ローリング---
タブ (航空機)---
スラット---
滑車---Pulley
ロッド---Rod
油圧---Hydraulic drive system
油圧サーボ (存在しないページ)---
アクチュエータ---Actuator
サーボ・バルブ (存在しないページ)---
フライ・バイ・ワイヤ---Aircraft flight control system
電線---Electrical cable
電気信号---
補助翼---Aileron
ヒンジ---Hinge (disambiguation)
モーメント---Moment (physics)
垂直尾翼---Vertical stabilizer
水平尾翼---Tailplane
方向舵---Rudder
昇降舵---Elevator (aeronautics)
モーメント---Moment (physics)
エンテ型飛行機---
スタビライザー---Stabilizer
第一次世界大戦---World War I
タブ (航空機)---
衝撃波---Shock wave
エアバスA380---Airbus A380
航空用エンジン---Aircraft engine
レシプロエンジン---Reciprocating engine
ディーゼルエンジン---Diesel engine
ガソリンエンジン---Petrol engine
ピストン---Piston
ガスタービンエンジン---Gas turbine
ジェットエンジン---Jet engine
減速機---Reduction drive
プロペラ---Propeller (aeronautics)
~
~
~
~
```

Figure 4.4: Glossary example for the topic “Airplane”.

Data is a precious thing and will last longer than the systems themselves.

Tim Berners-Lee

5 Data Preparation

5.1 Cleaning and Pre-processing Data

The output of the text scraping from the Python *Wikipedia* library is already preselected data, and even though the scraping process performs well getting only text, the data needs to be cleaned before it is sent to the next process-

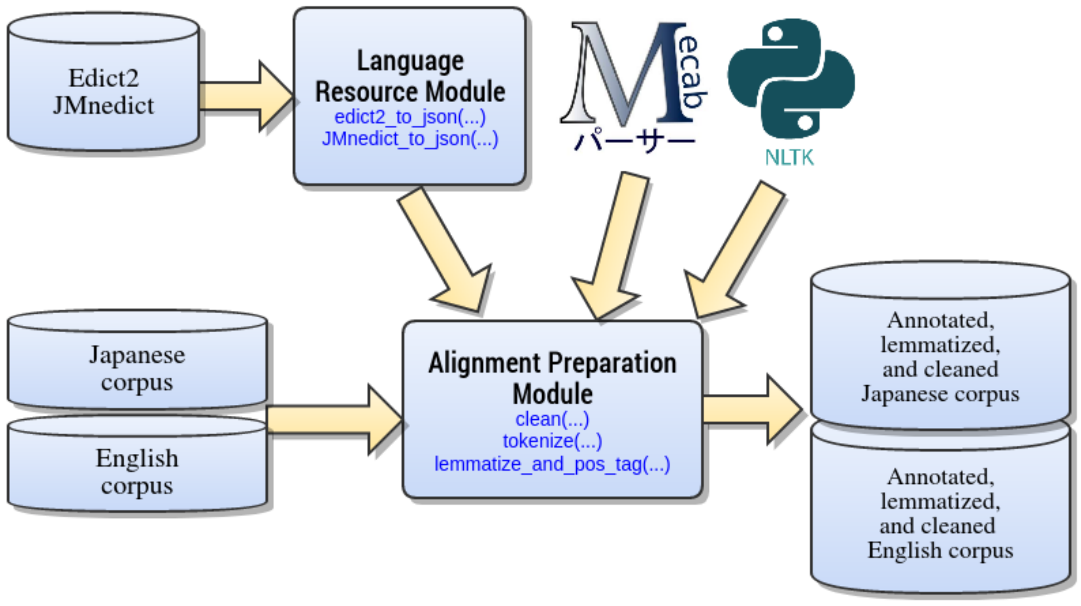

Edict2
JMnedict
Language Resource Module
edict2_to_json(...)
JMnedict_to_json(...)
Mecab
パーサー
NLTK
Japanese corpus
English corpus
Alignment Preparation Module
clean(...)
tokenize(...)
lemmatize_and_pos_tag(...)
Annotated, lemmatized, and cleaned Japanese corpus
Annotated, lemmatized, and cleaned English corpus

Figure 5.1: Modules of the Data Preparation Stage for cleaning the English and Japanese text collections and preparing them for alignment. Module names are written in bold face, function names (without parameters) are shown in small, blue font.

ing step. Apart from cleaning the data, we prepare the data by tokenizing, lemmatizing, and part-of-speech tagging (PoS tagging). Figure 5.1 shows an overview of the modules, functions, external resources, tools, and data used in this stage. Code 5.1 lists the libraries which are required for the rest of the stage. In addition to the libraries already discussed in Chapter 4, we also use the sentence tokenizer `sent_tokenize` from *nltk.tokenize*, the `WordNetLemma-tizer` class from *nltk.stem*, and the *wordnet* library from *nltk.corpus*. The use of these libraries is explained in Section 5.2.1.

```python
import os
import codecs
import re
import nltk
from nltk.tokenize import sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

Code 5.1: Preamble: Libraries for Data Preparation Stage.

5.2 Implementation of the Data Preparation Stage

5.2.1 Alignment Preparation Module

The first step is tokenizing on a sentence level in function **`tokenize`**, shown in Code 5.2.

Before that, however, we have to edit the data, due to the following issue: Since the text output from the *Wikipedia* library sometimes does not insert spaces between sentences, using the *nltk* sentence tokenizer function (Code 5.2:4,11) sometimes results in mistakes. In order to avoid this problem a small regular expression script is used to add a white space after the full stop at the end of a sentence. This is done in the command line input of the *vim* editor as shown in Code 5.3.

The `%` applies the following command to every line of the file, the `s` is the

```python
def tokenize():
    with codecs.open('corpus/text_english.txt','r','utf-8')as f:
        lines=f.read()
    sentences = sent_tokenize(lines)
    print(len(sentences))
    with open('corpus/tokenized_text_english.txt','w') as f:
        for sentence in sentences:
            f.write(sentence.encode('utf-8')+'\n')
    with open('corpus/text_japanese.txt','r')as f:
        lines=f.read()
    sentences=sent_tokenize(lines.decode('utf-8'))
    print(len(sentences))
    with open('corpus/tokenized_text_japanese.txt','w') as f:
        for sentence in sentences:
            f.write(sentence.encode('utf-8')+'\n')
```

Code 5.2: Function `tokenize`: Sentence tokenizing with NLTK.

```
:%s/\(\.\)\([A-Z]\)/\1 \2/g
```

Code 5.3: Vim command line: Adding white space after sentences.

substitute command. The substitute command has two parts separated and enclosed by three slashes, `/original/substitute/`. The first part is, in this case, identifying the punctuation mark between sentences. The period between the set of parentheses is the full stop of each sentence followed by the regular expression `[A-Z]` being any capital letter. With a small fraction of exceptions, this is a very easy way to identify the end of a sentence as opposed to a period after an abbreviation, a numbering, or special expression involving a period, such as "...". The second part, i.e. the substitute follows the second slash. This part refers to the period and to the capital letter found in the first part by \1 and \2, respectively. The white space between these two adds a white space between every instance of this pattern in the edited text. The backslashes used in front of the period, the parentheses and the numbers are necessary to escape the alternative functions of the special characters. A g character at the end of the substitute stands for "global", which means every instance of this pattern in a line will be processed.

Once the sentences are separated, we remove short sentences, i.e. sentences below a character count of 30 for English, and 20 for Japanese in function **clean**, shown in Code 5.4.

```python
def clean():
...
    for line in text_ja:
        if not re.search(r'^==.*',line):
            if len(line)>20:
                f_out_ja.write(line)
    for line in text_en:
        if not re.search(r'^==.*',line):
            if len(line)>30:
                f_out_en.write(line)
...
```

Code 5.4: Function `clean`: Removing unwanted data.

We found these character counts to be practical to remove headlines, and other short texts, which are not full sentences. The Japanese character count is lower since Japanese characters denote more content with fewer characters. The character counts certainly depend on the text type and other factors and should be set as needed. The 20/30 character cutoff proved to be sufficient to eliminate very short sentences, titles and headers in our experiments without removing good data. This is shown in Code 5.4. This function is also setup to remove any other noisy data. More searches can be included analogous to Code 5.4:4,8 if any other sentences with noisy patterns are to be identified and eliminated.

Next, we use the **`lemmatize_and_pos_tag`** function to lemmatize and PoS tag the data. First, we lemmatize the English text and convert each word to lower case, unless it is a proper noun. Lemmatized forms are crucial for finding translations during the Sentence Alignment Stage, and making proper nouns easy to identify enables an easy look up of such signal words in the dictionary. The step of lemmatizing is particularly computationally intensive; 100k English sentences took roughly 27 hours in the course of the first experiment. The hardware specifications of the desktop computer on which this runtime was observed is listed in Table 3.3, Section 3.4. Code 5.5 shows this

A recent study found periodic eye movements in the central bearded dragon of Australia, leading its authors to speculate that the common ancestor of amniotes may therefore have manifested some precursor to REMS.
Sleep deprivation experiments on non-human animals can be set up differently than those on humans.
The "flower pot" method involves placing a laboratory animal above water on a platform so small that it falls off upon losing muscle tone.
The naturally rude awakening which results may elicit changes in the organism which necessarily exceed the simple absence of a sleep phase.
This method also stops working after about 3 days as the subjects (typically rats) lose their will to avoid the water.

Table 5.1: English sentences, before lemmatizing.

process in detail. First, we word tokenize the text in Code 5.5:6, then PoS tag in the following line. By default the PoS format is a tuple with the word at position `[0]` and the PoS tag at position `[1]`. Starting at Code 5.5:11, we leave the entry in upper case if it is a proper noun, and convert to lower case if it is any other noun form or at the beginning of the sentence. If it is a verb (Code 5.5:18), we convert it to its dictionary form.

An example of five sentences before this processing is shown in Table 5.1, and the resulting changes in Table 5.2.

The Japanese text is also PoS tagged, however, this is done with an external command line tool, the *MeCab* open-source segmentation library, originally

```python
def lemmatize_and_pos_tag(lang):
...
        wl=WordNetLemmatizer()
        with codecs.open('corpus/\
   english_sentences_clean_lemmatized.txt','w', 'utf-8') as f:
            for line in lines:
                text=nltk.word_tokenize(line) # tokenize words
                pos=nltk.pos_tag(text) # PoS tagging
                sentence=''
                for word in pos:
                    lemma=word[0]
                    if word[1] != 'NNP': # if not proper noun
                        lemma=lemma.lower() # to lower case
                    if 'NN' in word[1]:  # if other noun
                        if 'NNS' in word[1]: # to lower case
                            lemma=lemma.lower()
                        # lemmatize nouns
                        lemma=wl.lemmatize(lemma,wordnet.NOUN)
                    if 'VB' in word[1]:
                        # lemmatize verbs
                        lemma=wl.lemmatize(word[0],'v')
                    sentence+=lemma+' '
                f.write(sentence+'\n')
```

Code 5.5: Function `lemmatize_and_pos_tag`: PoS tagging and lemmatizing English sentences.

a recent study find periodic eye movement in the central beard dragon of Australia , lead its author to speculate that the common ancestor of amniote may therefore have manifest some precursor to REMS .
sleep deprivation experiment on non-human animal can be set up differently than those on human .
the “flower pot” method involve place a laboratory animal above water on a platform so small that it fall off upon lose muscle tone .
the naturally rude awaken which result may elicit change in the organism which necessarily exceed the simple absence of a sleep phase .
this method also stop work after about 3 day as the subject (typically rat) lose their will to avoid the water .

Table 5.2: English sentences, after lemmatizing.

developed at the Nara Institute of Science and Technology. More information on *MeCab* is provided in Section 3.3.

Before that, however, we make sure to convert numerals in the Japanese text from full-width to half-width. Half width is used in English texts, hence comparisons will be much easier later. This is important, since numbers are particularly good signal tokens when aligning.

This is once more done with a regular expression in the vim editor command line as seen in Code 5.6. Similar to Code 5.3, it is a substitution com-

```
:%s/[\uff01-\uff5e]/\=nr2char(char2nr(submatch(0))-65248)/g
```

Code 5.6: Vim command line: Converting numbers from full-width to half-width representation.

mand. The first part, the text pattern to be replaced is the character range with the hexadecimal identification (unicode code point): `\uff01-\uff5e`. If a character is found matching this value, we convert it into its decimal representation with `char2nr`. We then subtract 65248 from it and convert it back into the code point representation with `nr2char`, which is the half-width code point of the same number.

5.2.2 Language Resource Module

The second part of this stage is the Language Resource Module. Here we transfer the dictionary files *JMnedict* and *edict2* to a JSON format. We chose to do this conversion for consistency, as we use the same JSON format for other resources, and for flexibility, as it allows the selection and addition of certain features from the resources. In Code 5.7 the conversion of *JMnedict* with function **`JMnedict_to_json`** is shown. In Code 5.7:10 we extract the named entities in Japanese and their English equivalents and store them as JSON objects.

In the function **`edict2_to_json`**, shown in Code 5.8, the conversion from the *edict2* file is shown, which is not as well structured as the XML-based *JMnedict*, therefore, we had to apply several string operations (Code 5.8:8-17)

```python
def JMnedict_to_json():
    with codecs.open('resources/JMnedict.xml','r','utf-8') as f:
        lines=f.readlines()
    dictJAEN={}
    kanji=''
    gloss=[]
    for line in lines:
        if '<entry>' in line:
            kanji=''
            gloss=[]
            glossitem=''
        if '<keb>' in line:
            kanji = line[5:-7] # get item (Japanese NE)
        if '<trans_det>' in line:
            # get glossary entry (English equivalent)
            glossitem = line[11:-13]
            gloss.append(glossitem)
            dictJAEN[kanji]=gloss # store in list
    with codecs.open('resources/JMnedict.json','w','utf-8') as f:
        json.dump(dictJAEN,f) # write to JSON file
```

Code 5.7: Function `JMnedict_to_json`: Converting *JMnedict* to JSON format.

to distill the content to our needs and finally also store it in JSON format.

```
def edict2_to_json():
...
    for line in lines:
        jap_word_list=[]
        jap_word_list=''.join(line.split(' ')[0]).split(';')
        # get translations
        translations=[]
        for idx, item in enumerate(line.split('/')):
            if idx!=0 and re.search('\w+',item) and 'EntL' not in \
   item:
                #removing {} and anything in between
                item=re.sub('\{[^)]*\}','',item).strip()
                #removing ()
                item=re.sub('\([^)]*\)','',item).strip()
                #removing ) -because of nested parenthesis mess in\
    edict2-
                item=re.sub('\)','',item).strip()
                if not re.search('![ - ~]',item) and not re.search\
   ('\?',item) and item!='':
                    translations.append(item)
        for item in jap_word_list:
            item=re.sub('\([^)]*\)','',item).strip() #removing ()
            dictJAEN[item]=translations
    with codecs.open('resources/edict2.json','w','utf-8') as f:
        json.dump(dictJAEN,f)
```

Code 5.8: Function `edict2_to_json`: Converting *edict2* to JSON format.

It's hardware that makes a machine fast. It's software than makes a fast machine slow.

Craig Bruce

6 Sentence Alignment

In this chapter we describe our approach to aligning Japanese and English sentences from a dataset which contains a mix of translations and paraphrasing; generally speaking sentences with various levels of similarity.

In order to align these sentences, various properties of text can be considered as indicators for equivalence. Some are mostly independent of language, such as sentence length, numbers, international terminology, or names. However, some of these comparisons are not always possible, due to various types of differences between highly dissimilar language pairs. The Japanese-English language pair is in this category and poses multiple issues, as mentioned in Section 1.1.3.

Other alignment criteria are content words, which have to be translated, in order to find their equivalents. Some word categories can and should be omitted, since they do not significantly contribute to finding equivalences but rather introduce noise, due to differences between the languages.

There are several advantages of translating from Japanese to English and to make English the language of comparison:

- English is written with one alphabet as opposed to Japanese, which makes comparison easier.
- It is easier to identify foreign words and terminology in Japanese, since they are usually written in Katakana, so they can be spotted prior to translation.
- English is well suited as a pivot language, so modules can be adjusted for other languages in combination with English.

The following section elucidates the procedure of assigning the alignment scores to sentences followed by a detailed discussion of the metric itself. Further, an example of the alignment of one Japanese sentence is broken down as a showcase. In this example, five sentences represent the set of alignment candidates in the English corpus. The chapter concludes with an explanation of the source code in the Sentence Alignment Stage.

6.1 Algorithm

The algorithm iterates over each word, i.e. POS entity, of a Japanese sentence and looks for equivalents within the English corpus. Particles and auxiliary verbs are omitted for the following reasons: They are frequent in Japanese and are less significant for the purpose of the alignment comparison than content words, numbers, named entities, and foreign words.

Examining sentence structures according to particles and auxiliary verbs would be possible, however it would require a deep analysis of the Japanese sentence and an equally deep analysis of the English sentence, in order to find a matching phrase. Additionally, varying uses of these particles can also lead to false assumptions and false matches. This is discussed in more detail in Section 1.1.3 on language specific issues. Although a detailed analysis of particles and phrases might benefit the alignment, the gain versus cost ratio would be disproportionally high compared to the ratio of simpler comparisons.

The algorithm sequence starts with a check for occurrences of numeric values written in Arabic numbers. This could be a measurement, a date, a quantity, etc. Since Japanese also uses Arabic numbers, this is one of the easiest and straightforward indicators for similar content. Each time a number value equivalent is found in the English corpus, the score of the particular sentence is increased.

The next step is finding the translation of each word in the Japanese sentence into English with the help of language resources like *edict2* (all resources used for translation and comparison are discussed in Section 3.2). As it is usually the case in translations, polysemy is the biggest problem here. A selection of the correct translation would require a semantic analysis of the entire sentence, or maybe even paragraph, hence is not possible, given the computational restrictions of this approach. In previous work by Utiyama and Isahara (2003) only two English translations were selected, using a simple heuristics based on frequencies of English words. In contrast to that, *every* available word in the list of possible translations is used in this algorithm. This increases the computational effort, however, it also improves accuracy significantly, so the performance loss is justified.

The English corpus is searched for a translation of each Japanese word. Special cases are numbers, which could be dates, measurements or quantities and named entities. Numbers are matched directly, while equivalents of named entities are obtained from the *JMnedict* language resource.

Each possible translation obtained from the dictionary resources is matched to candidates in the English corpus and an alignment score is increased, if a match is found. The sum of the match increments results in the alignment metric described in the next section. After the iteration of a Japanese sentence is complete, the English candidate sentence, i.e. the sentence with the highest score, is marked as a potential parallel sentence.

6.2 Alignment Metric

The alignment metric depends on the number of matches identified in the above described algorithm. It is a sum of matches, weighted and normalized over the length of the matched sentence:

$$score_i = \sum_{j=1}^{n_i} m_{i_j} \left(w + \frac{1}{l_i} \right)$$

$score_i$ ist the score of the i^{th} target sentence. n_i is the length of the source sentence, in this case the Japanese sentence, m_{i_j} is the match value of this token, in this case 1 for each match and 0 (zero) for no match. w is the weight, which can be adjusted according to how strong either a translation, a named entity or a numerical value match are to be emphasized as an alignment indicator. This value is currently a constant, since it is not changed for either of the above mentioned cases. However, the module allows for an easy change of the weight for each case to examine a potential improvement of the results. This is planned to be done in future work. Finally, l_i is the length of the target sequence, in this case, the English sentence.

6.3 Example of Sentence Alignment

The alignment process iterates over the corpus of Japanese sentences. For each sentence, the English sentence corpus is examined for similarities and a score is assigned for each sentence based on the metric described above.

For the purpose of illustration, one Japanese sentence and five English sentences represent the corpora. This would be the iteration of comparing one Japanese sentence from the Japanese corpus to the sequence of sentences in the English corpus, which in this example would consist of five sentences. The Japanese sentence is shown in Figure 6.1.

The five English alignment candidate sentences are shown in Figure 6.2.

2003 - 毎年どれだけの量の新たな情報が生み出されているかを見積もる試み

Figure 6.1: Japanese example sentence.

These sentences are processed and prepared as described in Section 5.2.1.

1. 2003 an attempt to estimate how much new information is created each year
2. In 2003, the United States invaded Iraq despite failing to pass a UN Security Council resolution for authorization.
3. A 2003 study argues the common chimpanzee should be included in the human branch as Homo troglodytes.
4. The Commonwealth's current highest-priority aims are on the promotion of democracy and development, as outlined in the 2003 Aso Rock Declaration.
5. An attempt to renew these efforts has be undertaken yearly, since 2003.

Figure 6.2: English example sentences.

As explained in Section 5.1, the Japanese sentence is tokenized and PoS-tagged with *MeCab*. The result is shown in Table 6.1.

token	POS	sub1	root-form
2003	名詞 (meishi=noun)	数 (kazu=number)	*
-	名詞 (meishi=noun)	サ変接続 (sahensetsuzoku =connection)	*
毎年 (maitoshi)	名詞 (meishi=noun)	副詞可能 (fukushikano=adverb)	毎年 (maitoshi)
どれ (dore)	名詞 (meishi=noun)	代名詞 (daimeishi=pronoun)	どれ (dore)
だけ (dake)	助詞 (joshi=particle)	副助詞 (fukujoshi=adjunct particle)	だけ (dake)
の (no)	助詞 (joshi=particle)	連体化 (rentaika=integration)	の (no)
量 (ryō)	名詞 (meishi=noun)	一般 (ippan=general)	量 (ryō)
の (no)	助詞 (joshi=particle)	連体化 (rentaika=integration)	の (no)
新た (arata)	名詞 (meishi=noun)	形容動詞語幹 (keiyōdōshigokan=adjective verb stem)	新た (ta)
な (na)	助動詞 (jodōshi =auxiliary verb)	*	だ (da)
情報 (jōhō)	名詞 (meishi=noun)	一般 (ippan=general)	情報 (jōhō)
が (ga)	助詞 (joshi=particle)	格助詞 (kakujoshi=case particle)	が (ga)
生み出さ (umidasa)	動詞 (dōshi=verb)	自立 (jiritsu=independent)	生み出す (umidasu)
れ (re)	動詞 (dōshi=verb)	接尾 (setsubi=suffix)	れる (reru)
て (te)	助詞 (joshi=particle)	接続助詞 (setsuzokujoshi=connective particle)	て (te)
いる (iru)	動詞 (dōshi=verb)	非自立 (jiritsu=independent)	いる (iru)
か (ka)	助詞 (joshi=particle)	副助詞／並立助詞／終助詞 (fukujoshi/heiritsujoshi/shūjoshi =adjunct/parallel particle/final particle)	か (ka)
を (wo)	助詞 (joshi=particle)	格助詞 (kakujoshi=case particle)	を (wo)
見積もる (mitsumoru)	動詞 (dōshi=verb)	自立 (jiritsu=independent)	見積もる (mitsumoru)
試み (kokoromi)	名詞 (meishi=noun)	一般 (ippan=general)	試み (kokoromi)

Table 6.1: PoS-tags of Japanese example sentence.

The result of the iteration of the Japanese sentence checking all language resources, while omitting particles and auxiliary verbs is shown in Table 6.2. In Table 6.3, the English sentences in their lemmatized form are matched with the output from the iteration of the Japanese sentence. The matches are highlighted in Table 6.3. The first sentence contains 5 matching tokens, sentence four has 4, while the others have one each. A graphical depiction of the matching process is shown in Figure 6.3. The tokens which are colored red are discarded, the tokens with POS-tag results in green are looked up in the lexical resources and equivalent expressions are searched amongst the English sentences. Matches are connected with lines and highlighted.

The scores are calculated according to the alignment metric in Section 6.2, which results in:

- *Sentence 1*: Sentence length: 13, weight:0.5, matches:5
  $(0.5 + \frac{1}{13}) * 5 = 2.885$;
- *Sentence 2*: Sentence length: 18, weight:0.5, matches:1
  $(0.5 + \frac{1}{18}) * 1 = 0.556$;
- *Sentence 3*: Sentence length: 17, weight:0.5, matches:1
  $(0.5 + \frac{1}{17}) * 1 = 0.559$;
- *Sentence 4*: Sentence length: 22, weight:0.5, matches:1
  $(0.5 + \frac{1}{22}) * 1 = 0.545$;
- *Sentence 5*: Sentence length: 12, weight:0.5, matches:4
  $(0.5 + \frac{1}{12}) * 4 = 2.333$

token	translation
2003	2003
毎年 (maitoshi)	every year, yearly, annually
量 (ryō)	progress
新た (arata)	new, fresh, novel
情報 (jōhō)	information, news, intelligence, advices
生み出さ (umidasa)	to create, to bring forth, to produce, to invent, to think up and bring into being, to give birth to, to bear
見積もる (mitsumoru)	to estimate
試み (kokoromi)	attempt, trial, experiment, endeavour (endeavor), effort, venture, initiative

Table 6.2: Translations of tokens of Japanese sentence into English.

The sentence with the highest score is considered the most likely alignment candidate, in this case, Sentence 1. According to the needs, i.e. the type of text, the domain, etc., the weights of the metric can be optionally adjusted at each match, so that, e.g. numbers, names, signal words or terminology is considered more indicative of an alignment, and such a match will result in a higher score. The weight is kept constant for the experiments shown in the thesis. Adjustments of this value might result in improvements of alignment, although it requires some experimentation, so this is reserved for future work.

2003-毎年どれだけの量の新たな情報が生み出されているかを見積もる試み (2003-maitoshi dore dake no ryō no aratana jōhō ga umidasa rete iru ka wo mitsumoru kokoromi)
2003 an attempt to estimate how much new information be create each year .
Alignment Score: 2.885
in 2003 , the United state invade Iraq despite fail to pass a UN Security Council resolution for authorization .
Alignment Score: 0.556
a 2003 study argue the common chimpanzee should be include in the human branch as Homo troglodytes .
Alignment Score: 0.559
the Commonwealth 's current highest-priority aim be on the promotion of democracy and development , as outline in the 2003 Aso Rock Declaration .
Alignment Score: 0.545
An attempt to renew these efforts has be undertaken yearly , since 2003.
Alignment Score: 2.333

Table 6.3: Target sentences with matches highlighted, and their resulting scores.

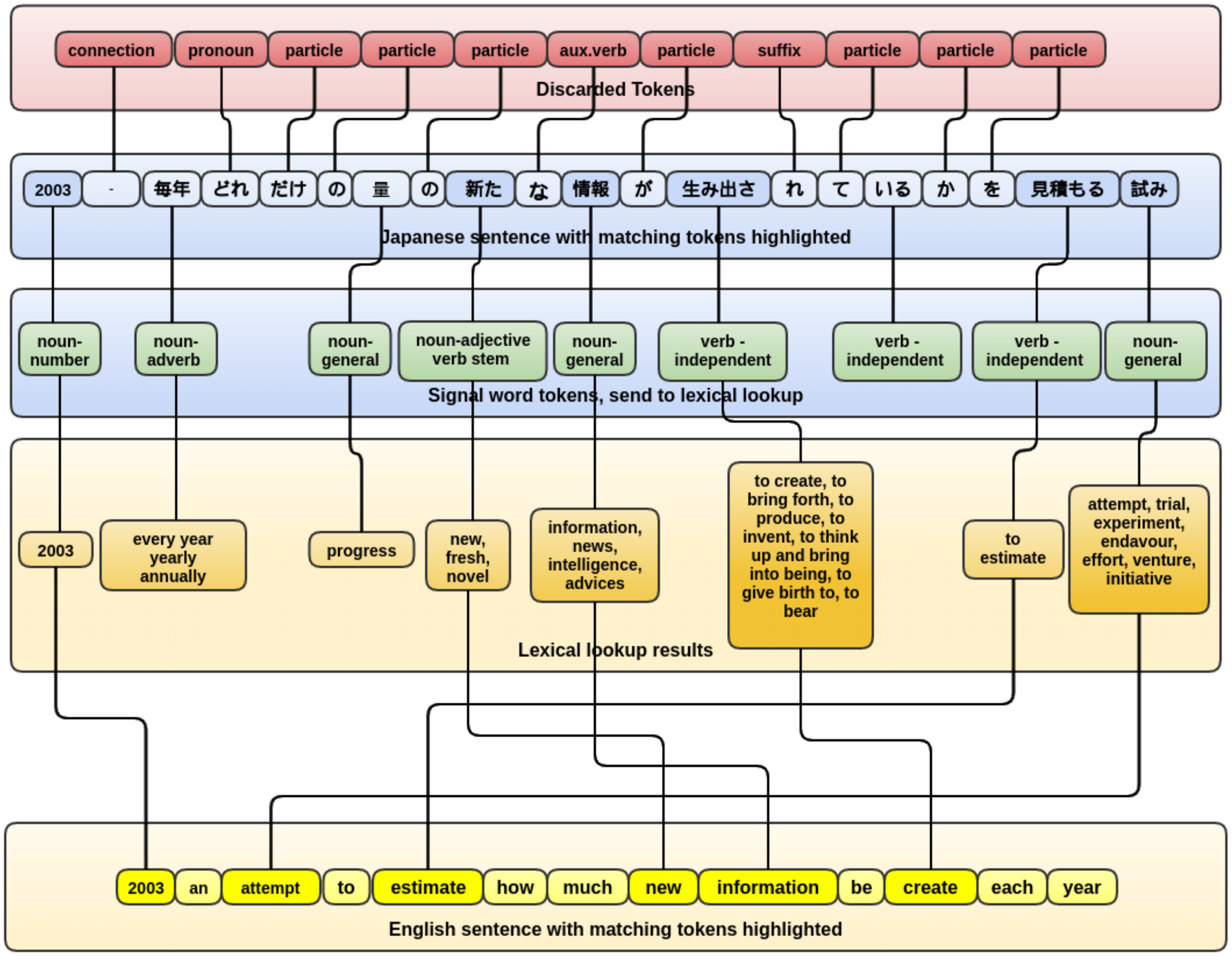


Figure 6.3: POS-tagging and matching visualized.

6.4 Implementation of the Sentence Alignment Stage

In this section, we explain the structure and source code of this stage as shown in Figure 6.4. The central part is the Alignment Module, which receives the prepared Japanese and English corpus data as well as dictionary data from *JMnedict* and *edict2* in JSON format, converted in the Language Resource Module in the Data Preparation Stage, described in Section 5.2.2. The output of this module are English and Japanese sentences in the form of a parallel corpus with each sentence having assigned a score according to the metric described in Section 6.2. For this stage we do not require any new library imports, other than some of the ones we already explained in the previous stages. We show a list of them in Code 6.1 for the sake of completeness and consistency.

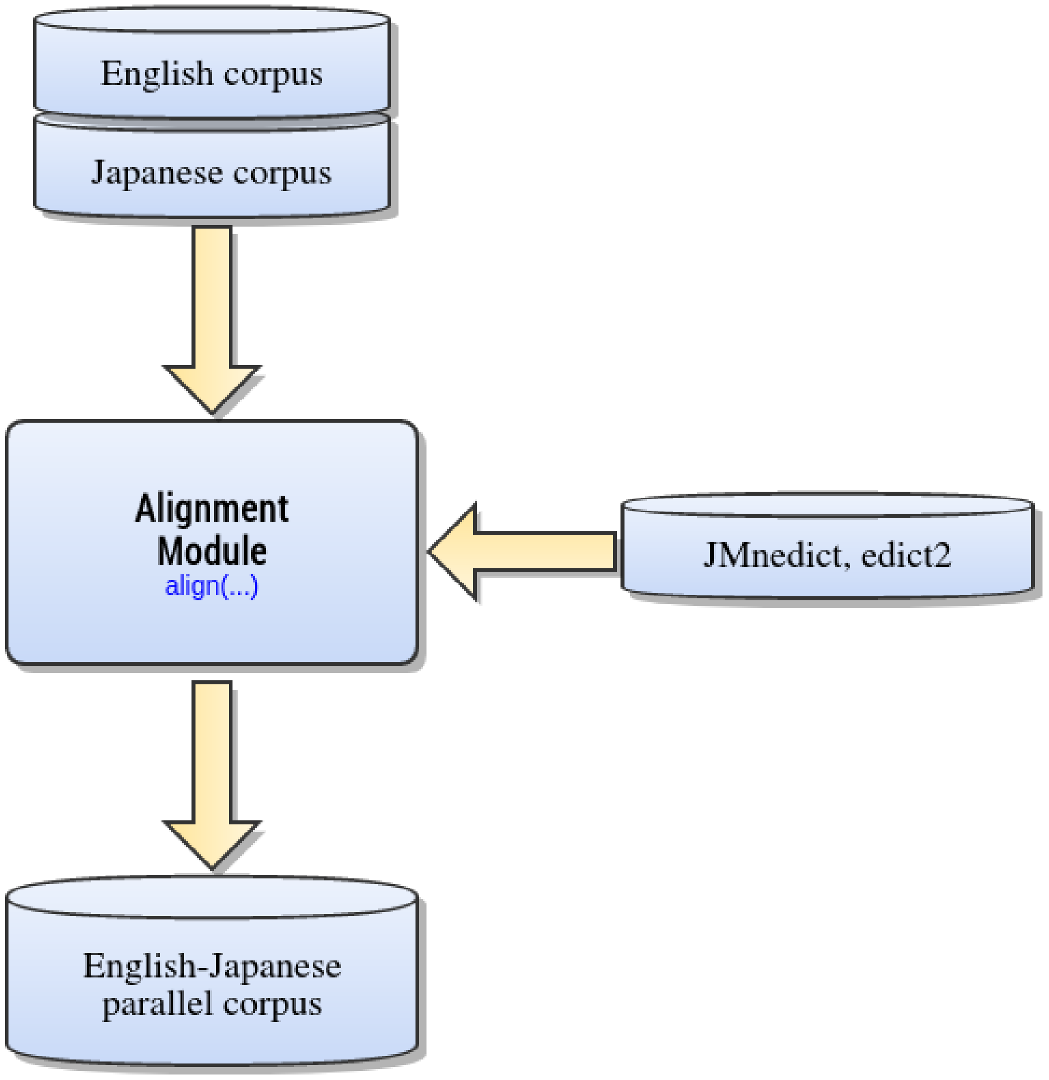


Figure 6.4: Overview of the Sentence Alignment Stage. Module names are written in bold face, function names (without parameters) are shown in small, blue font.

```python
import codecs
import json
import re
```

Code 6.1: Preamble: Libraries for Sentence Alignment Stage.

6.4.1 Alignment Module

This module aligns the English sentences with the best suited Japanese sentences according to criteria discussed in Section 6.1 and implemented in the **`align`** function.

In the first part of the function we load the JSON dictionaries prepared in the Data Preparation Stage (Code 6.2). It is important to remember the use of utf-8 encoding for reading and writing Japanese characters.

```python
def align():
    #dictionary files
    #**************************************************
    with codecs.open('JMnedict.json','r','utf-8') as f:
        JMnedict= json.load(f)
    with codecs.open('edict2.json','r','utf-8') as f:
        edict2= json.load(f)
    #**************************************************
```

Code 6.2: Function `align`: Loading dictionary files.

In the second part of the function, we initialize all variables and data structures that will be used (Code 6.3:1-6), in particular an array which will contain the scores for each alignment candidate sentence (6.4:7-9).

```python
    temp_lines = f_out.readlines()
    jap_counter=len(temp_lines)
    translations=[]
    match_dict={}
    content_matches_list=[]
    temp_counter=0
    array=[] # metric values will be stored here
    for idx in enumerate(eng_lemmas):
        # init to 0 for number of english sentences
        array.append(float(0))
```

Code 6.3: Function `align` (cont.): Initializing variables and data structures.

In Code 6.4:11, we start to iterate through the elements of the PoS tags of each Japanese sentence. The next line, although first in the code sequence, is the code block, which is executed when we reach the end of a Japanese sentence (Code 6.4:12). We explain this code block first even though it is not in line with the chronological program flow.

Each time every token of a Japanese sentence was traversed the result is written to two files. The first file (Code 6.4:16) contains the sequential number (in the input file) of the currently examined Japanese sentence and the sequential number of the English sentence, which was assigned the highest match probability, followed by the score. This file is written to allow for a quick overview during and after the alignment process and debugging, and

```
1  for idx in enumerate(eng_lemmas):
2      array.append(float(0))
3  for pos_tag in pos_tags:
4      if 'EOS' in pos_tag:
5          # weighted score algorithm
6          #*****************************************
7          # find highest value in <array> array, index is the \
   sentence number
8          f_out.write('<ALIGN><JAP>'+str(jap_counter)+'<ENG>'+str(\
   array.index(max(array))+1)+'<SCORE>'+str(max(array))+'\n')
9          f_parallel.write(str(max(array))+' ||| '+text_jap[\
   jap_counter-1].strip()+' ||| '+text_eng[array.index(max(array))\
   ].strip()+'\n')
10         #*****************************************
11         jap_counter+=1 # japanese sentence counter
12         for idx,val in enumerate(array):
13             array[idx]=float(0)
14         print 'Processing sentence '+str(jap_counter)+' of '+str(\
   len(text_jap))
```

Code 6.4: Function `align` (cont.): Writing parallel data and scores.

for shorter loading times in general. Especially when the data volume exceeds several dozens of Megabyte, a smaller representation of the results, which can be opened quickly, is very useful.

The file which contains the entire aligned data is created in the next line (Code 6.4:17). Here we write the score first, so it is visible immediately when the file is opened followed by the Japanese sentence, and finally the English alignment candidate.

The counter for accessing sentences in the data set is increased and the index for the array storing the alignment scores is reset for the iteration of the next Japanese sentence ((Code 6.4:19-22).

In the next part of the function shown in Code 6.5, we check for matches of numerals.

We make sure this check is performed only once per Japanese sentence with a temporary counter in Code 6.5:4-5, and determine whether there is a numeral in the sentence with a regular expression search in Code 6.5:7. If one or more numerals are found, the `for` loop at Code 6.5:10 is activated which iterates through the numerals in the Japanese sentence. Each one is then checked for a match in the English sentences. When a match is found, the score is increased, according to the metric (Section 6.2), in Code 6.5:17. The `try/except` blocks for both the regular expression check and the search for the match prevent the code from crashing if an out of bounds array member is referenced.

In the next part (Code 6.6), we take advantage of another potential signal word, the occasional use of Latin characters in Japanese.

This appears usually in acronyms and some foreign named entities. In

Code 6.6:3 we check for the *MeCab* class which identifies Latin characters. Numbers also fall in this category, however, MeCab tends to split numbers into individual numerals, rather than taking the entire number (for example, the year 2015 would be split into 2,0,1, and 5). This is the reason for dealing with numerals separately in Code 6.5. Hence, we skip further processing, if we encounter a number at this point. We then proceed to checking every English sentence for an occurrence of the Latin string we found in the Japanese sentence. If matches are found, the array with the scores is updated accordingly.

In Code 6.7 we check for matches from the named entity dictionary *JMnedict* and the dictionary *edict2*.

The named entity check is only performed, if MeCab has identified this token as a named entity (Code 6.7:3). If the current token is not a named entity we check for other excluded PoS tags in Code 6.7:15. A description of which tags are omitted and the reason behind it is provided in Section 6.1. All remaining PoS tags are checked in the dictionary.

In Code 6.8 we look for matches in English sentences, if a translation was retrieved from the dictionaries in Code 6.7. If a match is found, the score for that sentence is updated.

```python
digit_matches=[]
# do that only once per sentence (NB we're iterating POS tags!)
if temp_counter!=jap_counter:
    temp_counter=jap_counter
    try:
        # finding sequences of numerals
        digit_matches=re.findall(r'\d+',text_jap[jap_counter-1])
    except:
        pass
    # if numerals were found, looking for equivalents in english \
   data
    for match in digit_matches:
        jap_word=match
        eng_counter=0
        for eng_lemma in eng_lemmas:
            eng_counter+=1
            try:
                if re.search(r'\b'+jap_word+r'\b',eng_lemma):
                    array[eng_counter-1]+=float(0.5)+float(float\
   (1)/len(eng_lemma.split(' ')))
            except re.error,e:
                print 'passing re.error at translation: ', e
                pass
```

Code 6.5: Function `align` (cont.): Finding sentences with numerals.

```
# look for romaji in japanese sentences (works badly with
# numbers, since mecab tagging splits them up in single digits
if '名詞'.decode('utf-8') in pos_tag and not re.search(r'\d+',\
  jap_word):
   if re.search('\w',jap_word):
       eng_counter=0
       for eng_lemma in eng_lemmas:
           eng_counter+=1
           try:
               if re.search(r'\b'+jap_word+r'\b',eng_lemma):
                   array[eng_counter-1]+=float(0.5)+float(float\
  (1)/len(eng_lemma.split(' ')))
           except re.error,e:
               print 'passing re.error at translation: ', e
               pass
```

Code 6.6: Function `align` (cont.): Checking for Latin characters.

```
translations=[]
# if named entity, look for match in nedict
if '固有名詞'.decode('utf-8') in pos_tag:
    try:
        translations=JMnedict[jap_word]
    except KeyError,e:
        pass
else: # else look for match in regular dictionary
    # excluding certain japanese pos forms from dictionary lookup
    #(no particle, aux verb, etc)
    if not translations  and '助詞'.decode('utf-8') not in pos_tag\
     and '助動詞'.decode('utf-8') not in pos_tag:
        try:
            translations=edict2[jap_word]
        except KeyError,e:
            pass
```

Code 6.7: Function `align` (cont.): Dictionary lookups.

```
  if translations: # if any translation was found
    for translation in translations:
      eng_counter=0
      translation=re.sub('\(.*?\)','',translation)
      for eng_lemma in eng_lemmas:
        eng_counter+=1
        try:
          if re.search(r'\b'+translation+r'\b',eng_lemma\
  ):
            array[eng_counter-1]+=float(0.5)+float(\
  float(1)/len(eng_lemma.split(' ')))
            english_sentence_length=len(eng_lemma)
            japanese_sentence_length=len(text_jap[\
  jap_counter-1].encode('utf-8'))
        except re.error,e:
          print 'passing re.error at translation: ', e, \
  translation
          pass
  match_dict[jap_counter]=content_matches_list
f_out.close()
f_parallel.close()
```

Code 6.8: Function `align` (cont.): Finding translated matches.

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.

Stephen Hawking

7 Evaluation

The result of extracting, preparing, and aligning data in Chapters 4, 5, and 6 is a collection of Japanese-English sentence pairs. These sentence pairs are candidates for a parallel corpus. Whether they can be regarded as parallel sentences, i.e. translations which fulfil a certain quality measure, has to be evaluated. During the alignment process, each sentence pair is assigned a score, which indicates the similarities between said sentences. Hence the first evaluation is done automatically and inherently using the alignment metric.

7.1 Metric Score – Automatic Scoring

During the alignment process, a score is assigned according to the alignment metric. Roughly speaking, the higher the score, the higher the number of matches between these sentences. The quality measure given by the score for each alignment candidate results from the number of content words which are *potentially* equivalent. The possibility of additionally identifying wrong matches cannot be excluded, since every possible translation for ambiguous cases is considered and might result in false positives. On the other hand, there are matches that could be missed, such as numerals which are written as numbers in one language and words in the other.

Another factor, which influences the score, is the sentence length. The longer the sentence where a match was found, the lower the score. We do this under the assumption that longer sentences inherently have a higher chance to contain a certain word. The alignment metric adds the same weight to the score of the sentence pair where a match was found, which means that each match, regardless whether it is a content word, a number, terminology or a proper noun, is treated equally important. This weight can be adjusted, which is planned as future work, in order to examine alignments based on variable weighting. Details of the metric are discussed in Section 6.2. After the alignment is finished, the corpus is sorted by the score value. The best scoring 2000 sentences have been selected for evaluation by a translation expert.

7.2 Quality Measure by a Translation Expert

Evaluation of translations is a complex task and assigning a score can be very intricate depending on the quality requirements.

The evaluation of the data obtained by the framework is fairly straightforward as opposed to translations which require evaluation of a tone or a style. Nonetheless, the process of manually checking each sentence is laborious, hence we have selected a manageable number of sentences for the expert, based on our metric.

In order to identify potential translations as opposed to the sentence pairs which are not, but are still viable for statistical training and other language technology applications, three categories were used:

Sentence pairs that are not classified as translations can still be viable for statistical training and other language technology applications. We created categories that include these three possibilities.

- no equivalence,
- partially equivalent,
- good equivalence/potential translation.

The first category **“no equivalence”** includes sentence pairs that might contain matches, but apart from that the sentence cannot be classified as a translation, and therefore cannot be used as parallel data in any way. This is usually due to polysemy or other false positive alignment properties.

The second category **“partially equivalent”** includes sentence pairs containing only few correct matches between the sentences. Although they are not good translations, we can still use them for some language applications depending on the requirements.

The third category **“good equivalence/potential translation”** includes sentences where the majority of signal words match, or which can clearly be qualified as potential translations.

Quality amongst the potential translations is not scrutinized, since this is far beyond the scope of this work.

In other words, as described in Section 6.2, components of sentences which are equivalent and hence are indicative of a translation increase the match score. This metric is the first measure of how close the Japanese sentence is to the English sentence. Even if the score is high, this does not necessarily mean that the sentences are good translation candidates.

Depending on the requirements of the corpus, the quality criteria can be divided into several categories. An application which aims, for example, at terminology extraction, or requires a high number of equivalent content across sentences can purely rely on the similarity metric, described in Section 6.2. If the application requires aligned sentences that are translations or close enough to be considered as such, a more strict scoring needs to be applied. From this point on, a human expert is needed to confirm the quality.

7.3 Evaluation Results

In this section we summarize the output of our experiment and discuss our evaluation results. We created a corpus of 66,000 sentence pairs for evaluation. The 2000 highest scoring sentences, according to the alignment metric, were evaluated by a human expert. The expert scored the equivalence of the sentences with 1, 0.5, and 0, corresponding to the categories presented in Section 7.2. The percentage of good equivalence/potential translations was roughly 1% and partially equivalent 22.5%. The low ratio of good equivalents

score range	good equivalence/ potential translation	partially equivalent	no equivalence
9.5-10.0	5 %	29%	66%
9.0-9.49	4 %	25%	71%
8.5-8.99	4 %	21%	75%
8.0-8.49	3 %	20%	77%
7.5-7.99	1 %	10%	89%
7.0-7.49	2 %	15%	83%

Table 7.1: Results of human expert evaluation.

indicates that a large portion of Wikipedia articles is not a direct translation. The weight of the metric, which is used to generate the candidates, could be used as a tuning measure to yield potentially higher percentages. The result of the expert evaluation is summarized in Table 7.1.

As can be seen in Table 7.1, the percentage of good translation candidates increases with the score assigned by the alignment metric. This confirms the soundness of the metric. However, the overall result of good translations candidates is rather small. This could be due to false positive hits during the alignment (see Section 6.2 for a detailed discussion), and/or the fact that the English-Japanese Wikipedia content for the domain does not contain many translations, similar articles, and is more likely to have been created independently.

The second category, the sentence pairs which have been evaluated to be of similar content, also indicates the relation between the metric and human evaluation and is overall much higher. This can be taken as a rough indication of the percentage of similar English-Japanese content within this domain, which might have been created independently, but describing similar topics.

7.4 Runtimes

One of the benefits of our framework is that it can produce quick results on regular hardware that can be found in almost anyone's home these days. Even a fairly modern laptop computer is enough to obtain a dataset in a reasonable amount of time. In addition to the capability of creating potentially large parallel text resources, this opens up the possibility for freelance translators to quickly obtain a collection of bilingual data or a glossary of terms in a certain domain. This also allows quick checks for availability of text on Wikipedia for a selected set of topics.

There are many possible applications and most importantly the application is scalable. We do not need a huge data set to get off the ground like most deep learning applications but can work with small datasets as well as large ones. As opposed to many other parallel data acquisition applications mentioned in Section 2.7, our framework is scalable and can be run on modest hardware. In this showcase, we use a rather dated piece of hardware and demonstrate an acceptable turnaround for a small data set. In Table 7.2, we show the runtime for each module of the framework and the corresponding output of data, if applicable (English output is denoted with 'en', Japanese with 'ja').

We do not list The Translation Module separately in Table 7.2 since it is not part of the sequential processing pipeline but is invoked on demand by the Topic Extraction Module, the Formatting Module, and the Matching Module, as shown in Section 4.3.2, and Figure 4.3.

We use a seed of 10 articles for this example and limit the number of collected links to 2,500 per article. It is important to mention that some modules of the framework are more influenced by the performance of the hardware than others. The timing showcase presented in Table 7.2 is done on the desktop computer which is described in Section 3.4. As can be seen in this experiment, the Topic Extraction, Alignment Preparation, and the Alignment Module require the most time.

In case of the Topic Extraction Module that is due to the online lookups in Wikipedia, which means that this depends rather on bandwidth than on the

Module	Time	Output
Data Extraction Stage		
Topic Extraction	26m52s	en: 801087 articles, ja: 56736 articles
Formatting	1m12s	–
Matching	0.01s	–
Comparison	2m31s	–
Text Extraction	18.1s	en: 2037 lines, ja: 2072 lines en: 85,804 tokens, ja: 75,393 tokens
Data Preparation Stage		
Alignment Preparation	59m24s	en: 3510 sentences, ja: 805 sentences
Language Resource	21s	ja: 1,088,944 dictionary entries
Sentence Alignment Stage		
Alignment	24m50s	805 aligned sentences
Total		
–	1h43m48s	805 aligned sentences

Table 7.2: Runtime showcase for a small dataset.

computational capability of the machine. One option to improve the runtime would be to download a Wikipedia dump and work offline. However we chose an online solution in order to be able to detect the latest changes in Wikipedia articles. As described in Section 1.1, we want to analyse the content of Wikipedia and that includes its dynamic nature. However, if required, the modular structure of the framework allows for a fairly straightforward adjustment to process offline Wikipedia dumps.

The Alignment Preparation and Alignment Module are not dependent on online lookups and will be processed faster on a more powerful hardware. Running the exact same dataset on the faster laptop computer, described in Section 3.4, we observed approximately 20% reduction in runtime for the Alignment Module and 17% reduction for the Alignment Preparation Module. The absolute measurements for these two modules can be seen in Table 7.3. Apart from CPU speed, the most important factor is access time on the hard disk drive. The laptop in our experiment is equipped with a solid state drive with much faster access times, being crucial when many lookup operations in different files have to be done.

Regarding file access times, the system might be implemented to run in memory, or with the use of a temporary file system that is loaded on initial-

Module	Time	Output
Alignment Preparation	51m05s	en: 3510 sentences, ja: 805 sentences
Alignment	20m20s	805 aligned sentences

Table 7.3: Comparison runtime on faster system.

ization, however, we would have to know ahead, whether size requirements can be met. Additionally, the storage of text files and JSON objects on HDDs is more transparent and maintainable in the long run, and processing workflows can be resumed days, weeks, or even months later without having to start the entire process from the beginning. Similar to the choice of online versus offline processing, we chose flexibility, transparency, and sustainability over speed.

Even in the current form, the framework is capable of quick turnarounds for sizable datasets, but it certainly has much potential for optimization and performance tweaks to be even better suited for large scale data harvesting.

In the future,

computers may weigh no more than 1.5 tonnes.

Popular mechanics, 1949

8 Conclusion

8.1 Summary

The goal of the research performed in this thesis has been to create a method for *automatic creation of domain-specific parallel corpora between dissimilar language pairs*. Even though other methods for automatic creation of such corpora exist, almost all current approaches use a black box deep learning approach. The quality of parallel data collected by neural methods is not always of high quality, and due to the black box approach it is difficult to improve. Further, a large scale acquisition of data without control and eventually human evaluation allows for the processing of erroneous data of potentially significant volumes, as pointed out in Section 1.1.1. Despite most of the examples and our entire experiment setup being limited to Wikipedia, it is fairly safe to extrapolate and generalize to other sources for the sake of this argument.

A transparent method, based on language resources and rules, is certainly more labor intensive measured in relation to the data that is created, but allows for specific changes depending on the requirements and the application. Additionally, such a method can be fused with state of the art neural network approaches as an enhancement, specifically to bootstrap systems when little training data is available.

In order to prove the benefits of such an approach, we created a *framework for parallel data extraction*. We chose English-Japanese, since it is a good representation of a dissimilar language pair in many regards, as described in Section 1.1.3. We limited the data source to Wikipedia pages, since it is well structured and a good source for text, for almost any topic in many languages. We point out the benefits and the limitations of Wikipedia in Sections 1.1.1 and 4.1. With the flexibility for adaptation to other languages in mind, we kept the framework as modular and generic as possible, and we point out the parts of the framework that need to be adjusted in order to be used with other language pairs.

We explain in detail the framework that we have programmed as a proof of concept and which we use to show our preliminary results.

As we stated in Chapter 1, we broke the problem down into several stages, which we addressed thusly:

- We have constructed an algorithm to efficiently obtain domain-specific parallel corpus candidates from English and Japanese Wikipedia pages.
- We present an algorithm for aligning these English-Japanese sentences pairs.
- We propose a metric to quantify the similarity between English and Japanese sentences.
- We have programmed a framework and used it to build a parallel corpus.
- We evaluated this data with the help of a translation professional.

The framework we have implemented is a *chain of modules, divided in stages*. In the Data Extraction Stage (Chapter 4) we obtain domain-specific text data. The domain of the data is determined by the user, who defines how broad or how limited the domain shall be by choosing the seed topics. The way the text is obtained is as selective as possible to minimize the data to contain only the most likely candidates, which will be passed on for further processing. In the process of collecting data we also create English-Japanese glossaries. This self-contained stage can be used to quickly obtain data for different seed topics, or create glossaries on demand. This is potentially a powerful tool for translators and interpretors, who need glossaries for certain domains as well as for research applications for which quick access to domain-specific data is needed.

In the Data Preparation Stage (Chapter 5) we clean and prepare the data for the Sentence Alignment Stage. The preparation of the data is highly language specific and also depends on the data collected in the previous stage. This is the reason why this stage is also self contained and can be adjusted as needed.

In the Sentence Alignment Stage (Chapter 6) we implement the alignment algorithm and use our metric to quantify the quality of the alignment. The alignment method is based on dictionary lookups and transparent rules. This *white box* approach makes it possible to trace the algorithm and make observations on the processed data, use it in a teaching environment or enhance other applications, such as word-embedding models.

Finally, as a result of experimental runs with the framework, we obtained *66,000 parallel sentence candidates*. We selected a subset of the best scoring sentences for expert evaluation (Chapter 7).

8.2 Observations

In the following sections we describe the observations we made while programming the framework and running the experiments. We present our interpretation of the intermediate results and describe issues and things to be aware of. This mainly concerns the first two stages, which we will discuss in the following two subsections.

8.2.1 Data Extraction Stage

We chose Wikipedia for harvesting parallel data largely due to its consistent structure, its article links, and the ease of text extraction. At the same time we tried to answer the question of how much of Wikipedia's multilingual content is translated, paraphrased, similar, or completely asymmetrical.

The results for the topic seeds used in the experimental setups of this dissertation showed a low number of equivalent sentences (Chapter 7). We assume that much of the content in our experimental seed topics was either created in parallel, or created independently. There is a strong indication that direct translations make up a fairly low percentage of the content.

Another interesting observation was how quickly topics from various domains diverged from the initial domain with increasing link distance. For example, the topic "Tokyo" very quickly diverged into other domains, while extracted articles starting with "Plane" seemed to stay in this general domain longer. This is something to keep in mind when highly domain-specific data is required. The framework is flexible enough to allow for restricting criteria in such a case.

Overall, according to the study conducted in this dissertation, we observed that Wikipedia is a good source for parallel corpus extraction, due to its well-organized structure and availability of many language pairs, but a fairly poor source when large volumes of good quality translations are required.

8.2.2 Data Preparation Stage

The experience we gathered from this part of the framework is that handling encoding, and full and half-width representation can be quite challenging and requires much attention to detail, especially for Japanese. There are several common encodings used in Japanese, such as *JIS*, *Shift-JIS*, *EUC*, and *Unicode*. While *Unicode* is commonly used for many other scripts being a widely used standard, many Japanese language resources are encoded differently. Dealing with language tools, such as parsers, is especially tricky when they require input in a different encoding and output text which cannot be displayed on a utf-8 console, or in a utf-8 file.

We encountered an interesting issue with the *nltk* sentence tokenizer. It could not properly find sentence delimiters, whenever there was no white space between sentences. Since that was the case with most of the data extracted in our framework, we used a script in the vim command line to resolve this issue (Section 5.2.1).

8.3 Contributions

8.3.1 Theoretical Contributions

A theoretical contribution of this thesis is the description of the framework, which we hope to have presented well enough to be recreated as needed, be it in the Python programming language, as it is implemented in this work, or any other language of the reader's choice. We hope that the descriptions of our experiences but also the mentions of shortcomings will be helpful in such a recreation.

We described the workflow of a comprehensive process that goes from an initial input of topics to an output of large volumes of parallel data, adhering to basic software engineering principles of modularity and transparency. We chose to divide the framework into self-contained parts to allow for a transparent and flexible workflow. This modularity, and flexibility is important for debugging, analysing data in the process of running the framework and transparent data storage and maintenance.

The Data Extraction Stage is tailored to data collection from Wikipedia, but should another source be required, a different implementation of this stage can be substituted; as long as the output data format is consistent, the other stages, Data Preparation, and Sentence Alignment can be used requiring only very minor adjustments. The Data Extraction Stage can also be used solely for the purpose of creating glossaries, in this case all unneeded modules can be decoupled within the stage, to improve performance. We apply the same principle within the stages dividing the workflow into modules. While functioning as a workflow with one input and one output, they also can be run individually. This is particularly useful when working on large volumes of data which can take hours or days. If the process is interrupted, it can be picked up without much time loss due to the modularity. In this case, we can easily decouple modules and continue the process where it was left off. Throughout the entire stage we have used variables for the languages, which allows for a quick adaptation to other languages, which is mentioned in more detail in Section 8.4.

Similarly, the Data Preparation Stage is partitioned into modules. One module is responsible solely for reading and preparing dictionary data into a uniform JSON format, the other for cleaning, tokenizing, and PoS tagging. The conversion of the dictionary files allows for an easy way to load the most recent and up-to-date resources into the framework. Should these dictionary resources have a different format in the future, an adjustment in this module will be enough to ensure compatibility. Therefore, this module serves as a language resource interface.

The Sentence Alignment Stage consists of one module. The metric for measuring the similarity between sentences is located here. This metric allows us to quantify the number of equivalent tokens in relation to their estimated similarity value and the length of the sentence. These values were estimated based on grammatical knowledge and adjusted empirically.

Another contribution lies in the potential use of this framework as an enhancement method for MT systems which need quick and domain-specific input of data. Our framework allows for fairly speedy turnaround times, quick analyses and adjustments.

8.3.2 Practical Contributions

The practical contributions of this thesis are the parallel corpus of 66,000 sentence pairs and a selection of 2,000 sentence pairs, which are evaluated by a translation professional.

Our open source framework can be used to obtain more domain-specific parallel data, and can be adjusted for other language pairs and other data sources.

We are convinced that this framework, with a few additions and adjustments to increase the ease of use, can be of great value for translation professionals, who can use it in addition to existing computer assisted translation and terminology management tools. In particular the glossary extraction, which is part of the first stage of the framework, can deliver quick and accurate domain-specific data.

Last but not least, we used parts of our framework to obtain data for the development of the *EU Council Presidency Translator*[*], as contribution to this international project funded by the European Union. This project was carried out in conjunction with the *CEF eTranslation* platform [†]. We provided terminology data with the help of our glossary extraction function and monolingual data with our Data Extraction Stage with some minor adjustments.

8.4 Application to Other Language Pairs

The framework is written for the Japanese-English language pair, however it is meant to be flexible enough for a fairly quick adaptation to other languages.

Such an adaptation to other language pairs brings with it an entire list of interesting research questions: How are other language pairs represented on

[*] `https://translate2018.eu/` (Last accessed in August 2020)

[†] `https://translate2018.eu/#/about` (Last accessed in August 2020.)

Wikipedia? How close is their content between languages? This method can be an efficient way to quickly check whether parts of Wikipedia are filled in with machine translated content. It can be used to examine the ratio of text for certain topics within a language pair.

That being said, the difficulty of adjustment to other language pairs depends on the languages. The framework was built with comparison of dissimilar languages in mind, one of them being written in the Latin alphabet.

Changing English to another European language, which uses the same alphabet is very straight forward providing there are adequate POS-taggers and lemmatizers available for this language.

In order to substitute Japanese with another language, slightly more adjustments have to be made. The input and output is prepared for wide-width characters, which are used for Chinese characters and various other scripts, but the alignment method uses language-specific properties of Japanese, such as omission of particles, therefore this would have to be adopted.

8.5 Publications Resulting from this Research

In the process of researching this topic we have published our preliminary results in three peer-reviewed conferences.

The first publication Wloka (2015) was in the early stages of the work, where we suggested several alignment techniques. This paper was the first step towards this thesis and laid many foundations in terms of selective crawling and the use of language resources for this task. We explored various alignment techniques including the utilization of the Moses SMT toolkit (Koehn et al., 2007) for an experimental PoS based cluster matching method. However, this method soon became obsolete with the quickly improving NMT methods and we decided to embark on a new course. While the quickly improving NMT methods were making huge strives in translation quality for many language pairs, they were black box, very data hungry, and still struggled with dissimilar languages. We therefore decided to find a method which is transparent and can produce results quickly, to potentially enhance data intensive MT with knowledge-based bootstrapping methods.

Another consideration was the source from which we will harvest the data for alignment. At this stage we examined Wikipedia and decided to focus on the Japanese-English language pair. We presented our findings in a talk at the 2nd East Asian Translation Studies Conference in Japan (Wloka, 2016), and gained valuable insights in the resulting discussions, which we incorporated into this thesis.

While making progress on the Data Extraction Stage of the framework, we noticed the opportunity to use this part of the framework for building a tool for quick access to domain-specific, Japanese-English glossaries. Since in the Data Extraction Stage, the selection of domain-specific articles is determined by the links within the articles, the list which comprises the pre-selection set is coincidently a good collection of domain-specific terms and their translations, taken from Wikipedia, rather than dictionaries. We published these findings in (Wloka, 2018).

8.6 Limitations

The limitations of the approach described and implemented in this thesis are mainly its computational requirements. Although the articles which are scraped to obtain text are selected to maximize the probability of similar sentences, the articles are not compared individually, which would increase computational efficiency, but rather wholistically. The reason behind this is the assumption that even though we have text from different topics, the pre-selection yields a set of related articles, which might contain similar sentences across article pages. The pre-selection by topic similarity limits the data volume, but the overall exponential increase in computational time makes this approach computationally expensive. In order to limit CPU-hours we avoid processing some of the grammatical structures which could contribute to marginal accuracy improvements. This is described in Chapter 6.

8.7 Future Work

During the process of creating a parallel corpus, examining different possibilities, and attempting to answer the research questions, many more follow-up and related questions became apparent. Certainly, additional iterations of crawling and aligning are necessary to further confirm and refine the results presented in this thesis.

An opportunity for future work which requires some adjustments to the framework, but is certainly very interesting, is the extension of the modules to other languages and examining ratios of similar Wikipedia content across other language pairs.

As mentioned in Section 8.2.1 the identification of topics in the Data Extraction Stage can still be extended and refined. Due to the transparent architecture, rules, a language model, or word embeddings could be added to ensure the identification of data within a certain domain.

Further, a refinement of the metric can be examined by adjusting the weight, according to word category or other syntactic or semantic factors. As described in 6.1 we set the weights of the alignment metric by estimate and experimental fine-tuning. In the future this could be improved by quantitative analysis of the text obtained in the Data Extraction Stage (Chapter 4). By knowing how often certain categories of words, named entities, numerals, etc. appear in the data, we can make more accurate estimates for the weight values of the metric. Additionally, an extension to consider more grammatical structures during the alignment would be possible, although very computationally intensive, as mentioned in Section 8.6.

Especially taking into account the significant progress of deep learning in NLP with pre-trained word embedding models, such as BERT (Devlin et al., 2018), and the immense data requirements of these systems, we are confident that knowledge driven and transparent systems, which can work with little data, will complement these data hungry, opaque, and brittle deep learning approaches by allowing for quick boot strapping and much needed transparency and adaptability.

Our modular software chain is built with such flexibility and interconnectedness in mind and we are confident that it will enrich current and future state of the art research.

References

Aker, A., Kanoulas, E., and Gaizauskas, R. (2012). A light way to collect comparable corpora from the web. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

Arnold, D., Balkan, L., Meijer, S., Humphreys, R., and Sadler, L. (1993). *Machine Translation: an Introductory Guide*. Blackwells-NCC, London.

Artetxe, M. and Schwenk, H. (2019). Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610.

Bañón, M., Chen, P., Haddow, B., Heafield, K., Hoang, H., Esplà-Gomis, M., Forcada, M. L., Kamran, A., Kirefu, F., Koehn, P., Ortiz Rojas, S., Pla Sempere, L., Ramírez-Sánchez, G., Sarrías, E., Strelec, M., Thompson, B., Waites, W., Wiggins, D., and Zaragoza, J. (2020). ParaCrawl: Web-scale acquisition of parallel corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, Online. Association for Computational Linguistics.

Braschler, M. and Schäuble, P. (1998). Multilingual information retrieval based on document alignment techniques. In *Research and Advanced Technology for Digital Libraries*, volume 1513 of *Lecture Notes in Computer Science*, pages 183–197. Springer Berlin Heidelberg.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Cakmak, M. T., Acar, S., and Eryigit, G. (2012). Word alignment for English-Turkish language pair. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

Cettolo, M., Girardi, C., and Federico, M. (2012). Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268, Trento, Italy.

Chen, Y. and Eisele, A. (2012). MultiUN v2: UN documents with multilingual alignments. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv*, 1810.04805 [cs.CL].

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, pages 138–145, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Espla-Gomis, M., Klubicka, F., Ljubesic, N., Ortiz-Rojas, S., Papavassiliou, V., and Prokopidis, P. (2014). Comparing two acquisition systems for automatically building an English-Croatian parallel corpus from multilingual websites. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

Feng, F., Yang, Y., Cer, D., Arivazhagan, N., and Wang, W. (2020). Language-agnostic BERT sentence embedding. *arXiv*, 2007.01852 [cs.CL].

Grimes, S., Peterson, K., and Li, X. (2012). Automatic word alignment tools to scale production of manually aligned parallel texts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

Hutchins, J. (2012). Machine translation: General overview. In Mitkov, R., editor, *The Oxford Handbook of Computational Linguistics*, chapter 27, pages 501–511. Oxford University Press.

Jiang, Z., El-Jaroudi, A., Hartmann, W., Karakos, D., and Zhao, L. (2020). Cross-lingual information retrieval with BERT. *arXiv*, 2004.13005 [cs.IR].

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2016). Google's multilingual neural machine translation system: Enabling zero-shot translation. *arXiv*, 1611.04558 [cs.CL].

Koehn, P. (2005). Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the Tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand. AAMT, AAMT.

Koehn, P. (2017). Neural machine translation. *arXiv*, 1709.07809 [cs.CL].

Koehn, P., Hoang, H., Birch, A., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kumano, T. and Tokunaga, H. T. T. (2007). Extracting phrasal alignments from comparable corpora by using joint probability SMT model. In *Proceedings of the 11th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 95–103.

Laranjeira, B., Moreira, V., Villavicencio, A., Ramisch, C., and Finatto, M. J. (2014). Comparing the quality of focused crawlers and of the translation resources obtained from them. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

Luong, M.-T. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv*, 1604.00788 [cs.CL].

Ma, X. (2006). Champollion: A robust parallel text sentence aligner. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, pages 489–492.

Morishita, M., Suzuki, J., and Nagata, M. (2019). JParaCrawl: A large scale web-based English-Japanese parallel corpus. *arXiv*, 1911.10668 [cs.CL].

Nakazawa, T., Yaguchi, M., Uchimoto, K., Utiyama, M., Sumita, E., Kurohashi, S., and Isahara, H. (2016). ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 2204–2208, Portorož, Slovenia. European Language Resources Association (ELRA).

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Pryzant, R., Chung, Y., Jurafsky, D., and Britz, D. (2018). JESC: Japanese-English Subtitle Corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Rapp, R. (1999). Automatic identification of word translations from unrelated English and German corpora. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 519–526, Stroudsburg, PA, USA. Association for Computational Linguistics.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

Salton, G. (1971). Experiments in automatic thesaurus construction for information retrieval. In *IFIP Congress (1)*, pages 115–123.

Schwenk, H., Chaudhary, V., Sun, S., Gong, H., and Guzmán, F. (2019). Wikimatrix: Mining 135m parallel sentences in 1620 language pairs from wikipedia. *arXiv*, 1907.05791 [cs.CL].

Sennrich, R. and Volk, M. (2010). MT-based sentence alignment for OCR-generated parallel texts. In *The Ninth Conference of the Association for Machine Translation in the Americas (AMTA 2010)*.

Smith, J. R., Saint-Amand, H., Plamada, M., Koehn, P., Callison-Burch, C., and Lopez, A. (2013). Dirt cheap web-scale parallel text from the common crawl. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1374–1383, Sofia, Bulgaria. Association for Computational Linguistics.

Tanaka, Y. (2001). Compilation of a multilingual parallel corpus. In *Proceedings of PACLING 2001*, pages 265–268.

Thompson, B. and Koehn, P. (2019). Vecalign: Improved sentence alignment in linear time and space. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1342–1348, Hong Kong, China. Association for Computational Linguistics.

Uszkoreit, J., Ponte, J., Popat, A., and Dubiner, M. (2010). Large scale parallel document mining for machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1101–1109, Beijing, China. Coling 2010 Organizing Committee.

Utiyama, M. and Isahara, H. (2003). Reliable measures for aligning Japanese-English news articles and sentences. In *Proceedings of the 41st Annual Meet-*

ing of the Association for Computational Linguistics, pages 72–79, Morristown, NJ, USA. Association for Computational Linguistics.

Varga, D., Halaácsy, P., Kornai, A., Nagy, V., Németh, L., and Trón, V. (2005). Parallel corpora for medium density languages. In *Proceedings of the RANLP 2005 Conference*, pages 590–596.

Vauquois, B. (1968). A survey of formal grammars and algorithms for recognition and transformation in mechanical translation. In *IFIP Congress (2)*, pages 1114–1122.

Weaver, W. (1949/1955). Translation. In *Machine Translation of Languages*, pages 15–23. MIT Press, Cambridge, MA. Reprinted from a memorandum written by Weaver in 1949.

Winiwarter, W. (2013). Mastering Japanese through augmented browsing. In *Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '13, pages 179:179–179:188. ACM.

Winiwarter, W. (2015). JILL: Japanese incidental language learning. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '15, pages 9:1–9:9. ACM.

Wloka, B. (2015). Towards automated creation of high quality domain-specific machine translation resources. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2015, Brussels, Belgium, December 11-13, 2015*, pages 38:1–38:4. ACM.

Wloka, B. (2016). Is Wikipedia a good candidate for Japanese-English bilingual corpus harvesting? In *Proceedings of the 2nd East Asian Translation Studies Conference*, Tokyo, Japan.

Wloka, B. (2018). Identifying bilingual topics in Wikipedia for efficient parallel corpus extraction and building domain-specific glossaries for the Japanese-English language pair. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Paris, France. European Language Resources Association (ELRA).

Zhang, Y., Vogel, S., and Waibel, A. (2004). Interpreting BLEU/NIST scores: How much improvement do we need to have a better system? In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, pages 2051–2054.

Zhao, S., Niu, C., Zhou, M., Liu, T., and Li, S. (2008). Combining multiple resources to improve SMT-based paraphrasing model. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1021–1029.

Zhu, J., Xia, Y., Wu, L., He, D., Qin, T., Zhou, W., Li, H., and Liu, T.-Y. (2020). Incorporating BERT into neural machine translation. *arXiv*, 1411.5595 [cs.CL].

Ziemski, M., Junczys-Dowmunt, M., and Pouliquen, B. (2016). The United Nations Parallel Corpus v1.0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3530–3534, Portorož, Slovenia. European Language Resources Association (ELRA).

A
Source Code

In this appendix, the source code of the program developed in the course of this thesis is presented in a modular, sequential fashion.

The overview in Figure A.1 shows the modules of the Data Extraction Stage of the framework. Figure A.2 gives an overview of the Data Preparation Stage.

Figure A.3 depicts the modules of the Sentence Alignment Stage.

All schematic depictions show which functions are being called from each module. The functions are listed below in sequential order. For the purpose of clarity, parameters of the functions are omitted in the overview and in the titles. All figures in the appendix are repeated and equivalent with the figures in the chapters above. They are included in the appendix for convenience and readability of the source code.

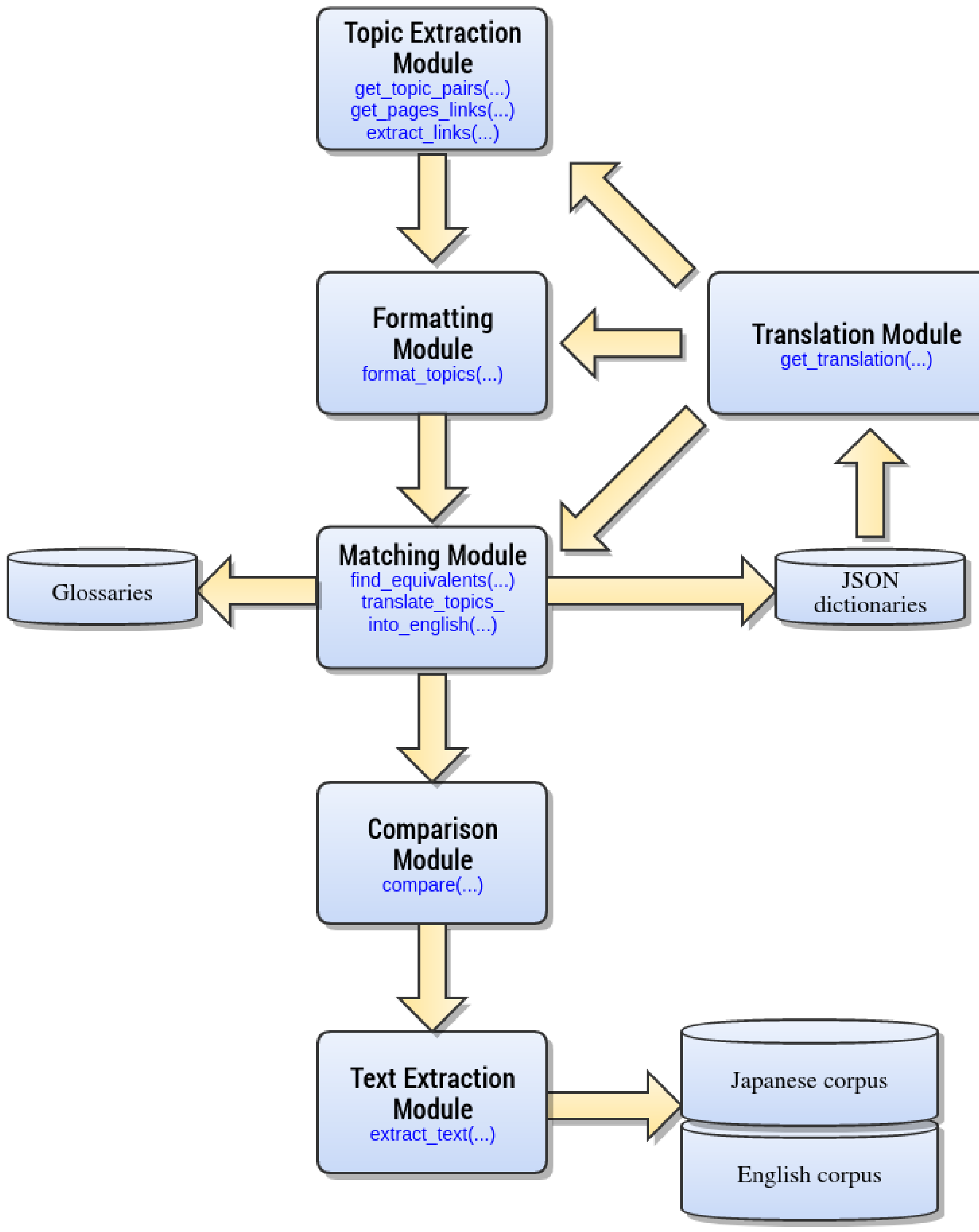


Figure A.1: Modules of the Data Extraction Stage for selective harvesting and text extraction from Wikipedia articles. The goal of this chain of modules is to obtain candidate sentences for a parallel corpus. Module names are written in bold face, function names (without parameters) are shown in small, blue font.

A.1 Data Extraction Stage

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from bs4 import BeautifulSoup
import requests
import codecs
import re
import json
import urllib
import os
```

A.1.1 Topic Extraction Module

Function – get_topic_pairs

```python
def get_topic_pairs(topic_list):
    # for each start topic
    for topic in topic_list:
        start_topic= topic.strip()
        #get start topic in japanese by ID check
        topic_ja = get_translation(start_topic,'ja','en')
        #open files to store subtopics
        ftopics_en = codecs.open('data/'+start_topic+'_topics_en.\
  txt','w', encoding='utf8')
        ftopics_ja = codecs.open('data/'+start_topic+'_topics_ja.\
  txt','w', encoding='utf8')

        #call get_pages function to get subtopics for english
        topics = get_pages_links(start_topic, 'en')
        #write results to file and close output file
        for topic in topics:
            ftopics_en.write(topic[0]+'->'+topic[1]+'\n')
        ftopics_en.close()

        #call get_pages function to get subtopics for japanese
        topics = get_pages_links(topic_ja, 'ja')
        #write results to file and close output file
        for topic in topics:
            ftopics_ja.write(topic[0]+'->'+topic[1]+'\n')
        ftopics_ja.close()
```

Function – get_pages_links

```
# return all links on the topic page and all subsequent links
def get_pages_links(topic, lang):
    start_url = 'https://'+lang+'.wikipedia.org/wiki/'
    domain = 'https://'+lang+'.wikipedia.org'
    start_url=start_url+topic #main topic link
    items = []
    # get own title, link titles and links for main topic
    title, ext_titles, ext_links = extract_links(url=start_url)
    # store in items list
    items.extend(zip([title]*len(ext_titles), ext_titles))
    for ext_link in ext_links:
        # omitting Wiktionary entries and pronunciation links
        if 'wikt' not in ext_link and 'Help:IPA' not in ext_link:
            try:
                # resolve encoding issues
                ext_link=urllib.unquote(ext_link).decode('utf-8')
                # get own title, link titles and links for main \
topic
                title, ext_titles, ext_links = extract_links(\
domain + ext_link)
                # store in items list
                items.extend(zip([title]*len(ext_titles), \
ext_titles))
```

```
            except UnicodeEncodeError, e:
                print('UnicodeEncodeError at: ',ext_link,'-reason:\
   ', str(e))
                pass
        if len(items) > 2500:
            break
    return items
```

Function – extract_links

```
# return a list of links to other Wikipedia articles
def extract_links(url):
    # get soup with lxml parser
    soup = BeautifulSoup(requests.get(url).content,'lxml')
    # find all the paragraph tags
    p_tags = soup.findAll('p')
    # gather all <a> tags
    a_tags = []
    for p_tag in p_tags:
        a_tags.extend(p_tag.findAll('a'))
    # filter the list : remove invalid links
    a_tags = [ a_tag for a_tag in a_tags if 'title' in a_tag.attrs\
    and 'href' in a_tag.attrs and not 'class' in a_tag.attrs]
    # get all the article titles
    titles = [ a_tag.get('title') for a_tag in a_tags ]
    # get all the article links
    links  = [ a_tag.get('href')  for a_tag in a_tags ]
    # get own title
    self_title = soup.find('h1', {'class' : 'firstHeading'}).text
    return self_title, titles, links
```

A.1.2 Translation Module

Function – get_translation

```python
# return the Wikipedia site equivalent in a target language
def get_translation(topic,source_lang,target_lang):
    # use Wikipedia's json database to look it up
    json_url='https://'+source_lang+'.wikipedia.org/w/api.php?\
   action=query&titles='+topic+'&prop=langlinks&lllimit=500&format\
   =json'
    content = requests.get(json_url).content
    json_data = json.loads(content)
    item=''
    # iterate through json hierarchy to find langlinks category
    try:
        for i in json_data["query"]["pages"]:
            pageid=i
    except KeyError, e:
        print('KeyError at topic:',topic,' - reason: ',str(e))
        pass
    except TypeError, e:
        pass
    try:
        for i in json_data["query"]["pages"][pageid]["langlinks"]:
            # in langlinks category, find desired langauge
            if i['lang']==target_lang:
                # there is the topic equivalent
                item = i['*']
    except KeyError, e:
        print('Keyerror at topic',topic,' - reason: ',str(e))
        pass
    except TypeError, e:
        pass
    return item
```

A.1.3 Formatting Module

Function – format_topics

```
def format_topics():
    file_list=[]
    os.chdir('./data')

    for file in glob.glob('*topics_en*.txt'):
        file_list.append(file)
    for file in file_list:
        with codecs.open(file,'r','utf-8') as f:
            previous_line=f.readline()
            f_temp=codecs.open('topics/'+previous_line.split('->')\
[0]+'_en.txt','w','utf-8')
            lines = f.readlines()
            counter=0
            for line in lines:
                counter+=1
                if line.split('->')[0]!=previous_line.split('->')\
[0]:
                    f_temp.close()
                    try:
                        f_temp=codecs.open('topics/'+line.split('\
->')[0]+'_en.txt','w','utf-8')
                    except IOError, e:
                        print('cannot open file',str(e))
                        break
                    previous_line = line
                f_temp.write(line.split('->')[1])


    file_list=[]
    for file in glob.glob('*topics_ja*.txt'):
        file_list.append(file)
    for file in file_list:
        with codecs.open(file,'r','utf-8') as f:
            previous_line=f.readline()
            print('translating '+previous_line.split('->')[0].\
encode('utf-8'))
            translation = get_translation(previous_line.split('->'\
)[0].encode('utf-8'),'ja','en')
            f_temp=codecs.open('topics/'+translation+'_ja.txt','w'\
,'utf-8')
            lines = f.readlines()
            for line in lines:
                if line.split('->')[0]!=previous_line.split('->')\
[0]:
                    f_temp.close()
                    previous_line = line
                    translation = get_translation(previous_line.\
split('->')[0].replace\
                        ('Wikipedia:','').encode('utf-8'),'ja'\
,'en')
                    f_temp=codecs.open('topics/'+translation+'_ja.\
txt','w','utf-8')
                f_temp.write(line.split('->')[1])
        f_temp.close()
```

A.1.4 Matching Module

Function – translate_topics_into_english

```
def translate_topics_into_english():
    file_list=[]
    os.chdir('.')

    # if there is no dictionary file, open a new one
    if not os.path.exists('./topics_dict_ja_en.json'):
        empty_dict={}
        f=open('topics_dict_ja_en.json','w')
        json.dump(empty_dict,f)
        f.close()
    topic_dict_ja_en={}
    with codecs.open('topics_dict_ja_en.json','r',encoding='utf8')\
     as fdict:
        topic_dict_ja_en=json.load(fdict)

    # translate sorted japanese files
    file_list=[]
    for file in glob.glob('data/topics/pairs/*_ja.txt'):
        file_list.append(file)
    for file in file_list:
        fout=codecs.open(file[:-4]+'_en_ja.txt','w','utf-8')
        with codecs.open(file,'r','utf-8') as f:
            lines=f.readlines()
            for line in lines:
                try:
                    fout.write(topic_dict_ja_en[line[:-1]]+'\n')
                except KeyError, e:
                    print('KeyError: ', str(e))
                    trans=get_translation(line[:-1],'ja','en')
                    fout.write(trans+'\n')
                    topic_dict_ja_en[line.rstrip()]=trans
                    print(line.rstrip()+'->'+trans+' added to \
   dictionary')
                    pass

    f=open('topics_dict_ja_en.json','w')
    json.dump(topic_dict_ja_en,f)
    f.close()
```

Function – find_equivalents

```python
def find_equivalents():
    file_list_ja=[]
    file_list_en=[]
    os.chdir('./data/topics') # change to data dir
    for file in glob.glob('*_ja.txt'): # for every japanese file
        file_list_ja.append(file[:-7]) # get topic from filename
    for file in glob.glob('*_en.txt'): # for every english file
        file_list_en.append(file[:-7]) # get topic from filename
        #store data in pairs
    for item in file_list_en:
        if item in file_list_ja:
            copy('./'+item+'_ja.txt','./pairs/'+item+'_ja.txt')
            copy('./'+item+'_en.txt','./pairs/'+item+'_en.txt')
            topic_pairs.append(item)
    os.chdir('../../') # back to main dir
```

A.1.5 Comparison Module

Function – compare

```python
def compare():
    file_list_ja=[]
    file_list_en=[]
    os.chdir(os.path.dirname(os.path.realpath(__file__)))
    os.chdir('./data/topics/pairs')


    for file in glob.glob('*_en.txt'):
        file_list_en.append(file[:-7])

    scounter=0
    similar=[]
    #open both files and compare
    for item in file_list_en:
        common_counter=0
        topics_ja=[]
        topics_en=[]
        with codecs.open(item+'_ja_en_ja.txt','r','utf-8') as fja:
            lines=fja.readlines()
            for line in lines:
                topics_ja.append(line.split())
        with codecs.open(item+'_en.txt','r','utf-8') as fen:
            lines=fen.readlines()
            for line in lines:
                topics_en.append(line.split())
        for topic_ja in topics_ja:
            for topic_en in topics_en:
                if topic_ja==topic_en:
                    common_counter+=1
                    break
        # calculating ratio, after counting extracted links for \
each topic
        if len(topics_ja)>0:
            scounter+=1
            similar.append(item)
            ratio = float(float(common_counter)/float(len(\
topics_ja)))
            if ratio>0.7: # ratio threshold value
                print str(common_counter)+' link matches in topic \
>'+ item + '< out of total '+ str(len(topics_en))+' links -> \
ratio: '+ str(round(ratio,3))
    print 'Total similar pages count: '+str(scounter)
    return similar
    os.chdir('../../../') # back to main dir
```

A.1.6 Text Extraction Module

Function – extract_text

```
def extract_text(link_list):
    os.chdir('./data/topics/pairs')
    ftext=codecs.open('text_english.txt','w','utf-8')
    for item in link_list:
        try:
            # passing article name to get reference to page
            p = wikipedia.page(item.strip())
            ftext.write(p.content) # getting text from Wikipedia \
   page
        except wikipedia.exceptions.WikipediaException as e:
            pass
        scounter-=1
    ftext.close()

    # same for Japanese
    ftext=codecs.open('text_japanese.txt','w','utf-8')
    for item in link_list:
        wikipedia.set_lang('ja')
        try:
            # passing article name to get reference to page
            p = wikipedia.page(get_translation(item.strip(),'en','\
   ja'))
            ftext.write(p.content)
        except wikipedia.exceptions.WikipediaException as e:
            pass
        scounter2-=1
    ftext.close()
    os.chdir('../../../') # back to main dir
```

A.2 Data Preparation Stage

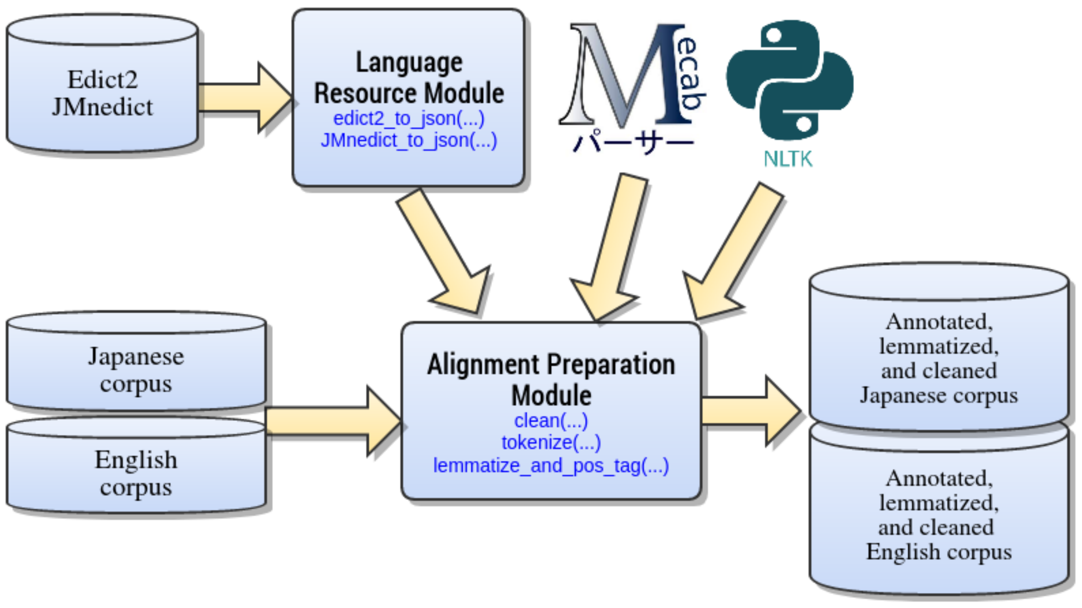

Edict2
JMnedict
Language Resource Module
edict2_to_json(...)
JMnedict_to_json(...)
Mecab
パーサー
NLTK
Japanese corpus
English corpus
Alignment Preparation Module
clean(...)
tokenize(...)
lemmatize_and_pos_tag(...)
Annotated, lemmatized, and cleaned Japanese corpus
Annotated, lemmatized, and cleaned English corpus

Figure A.2: Modules of the Data Preparation Stage for cleaning the English and Japanese text collections and preparing them for alignment. Module names are written in bold face, function names (without parameters) are shown in small, blue font.

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os
import codecs
import re
import nltk
from nltk.tokenize import sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

A.2.1 Alignment Preparation Module

Function – tokenize

```
def tokenize():
    with codecs.open('corpus/text_english.txt','r','utf-8')as f:
        lines=f.read()

    sentences = sent_tokenize(lines)
    print(len(sentences))

    with open('corpus/tokenized_text_english.txt','w') as f:
        for sentence in sentences:
            f.write(sentence.encode('utf-8')+'\n')

    with open('corpus/text_japanese.txt','r')as f:
        lines=f.read()

    sentences=sent_tokenize(lines.decode('utf-8'))
    print(len(sentences))

    with open('corpus/tokenized_text_japanese.txt','w') as f:
        for sentence in sentences:
            f.write(sentence.encode('utf-8')+'\n')
```

Function – clean

```
def clean():
    with codecs.open('corpus/tokenized_text_japanese.txt','r','utf\
   -8') as f:
        text_ja=f.readlines()
    with codecs.open('corpus/tokenized_text_english.txt','r','utf\
   -8') as f:
        text_en=f.readlines()
```

```python


    f_out_en = codecs.open('corpus/english_sentences_clean.txt','w\
   ','utf-8')
    f_out_ja = codecs.open('corpus/japanese_sentences_clean.txt','\
   w','utf-8')

    for line in text_ja:
        if not re.search(r'^==.*',line):
            if len(line)>20:
                f_out_ja.write(line)
    for line in text_en:
        if not re.search(r'^==.*',line):
            if len(line)>30:
                f_out_en.write(line)

    f_out_en.close()
    f_out_ja.close()
```

Function – lemmatize_and_pos_tag

```python
def lemmatize_and_pos_tag(lang):
    #for english
    if not lang or lang=='en':
        with codecs.open('corpus/english_sentences_clean.txt','r',\
   'utf-8') as f:
            lines = f.readlines()
        wl=WordNetLemmatizer()
        with codecs.open('corpus/\
   english_sentences_clean_lemmatized.txt','w', 'utf-8') as f:
            for line in lines:
                text=nltk.word_tokenize(line) # tokenize words
                pos=nltk.pos_tag(text) # PoS tagging
                sentence=''
                for word in pos:
                    lemma=word[0]
                    if word[1] != 'NNP': # if not proper noun
                        lemma=lemma.lower() # to lower case
                    if 'NN' in word[1]:  # if other noun
                        if 'NNS' in word[1]: # to lower case
                            lemma=lemma.lower()
                        # lemmatize nouns
                        lemma=wl.lemmatize(lemma,wordnet.NOUN)
                    if 'VB' in word[1]:
                        # lemmatize verbs
                        lemma=wl.lemmatize(word[0],'v')
                    sentence+=lemma+' '
                f.write(sentence+'\n')
```

A.2.2 Language Resource Module

Function – JMnedict_to_json

```
def JMnedict_to_json():
    with codecs.open('resources/JMnedict.xml','r','utf-8') as f:
        lines=f.readlines()
    dictJAEN={}
    kanji=''
    gloss=[]
    for line in lines:
        if '<entry>' in line:
            kanji=''
            gloss=[]
            glossitem=''
        if '<keb>' in line:
            kanji = line[5:-7] # get item (Japanese NE)
        if '<trans_det>' in line:
            # get glossary entry (English equivalent)
            glossitem = line[11:-13]
            gloss.append(glossitem)
            dictJAEN[kanji]=gloss # store in list
    with codecs.open('resources/JMnedict.json','w','utf-8') as f:
        json.dump(dictJAEN,f) # write to JSON file
```

Function – edict2_to_json

```
def edict2_to_json():
    with codecs.open('resources/edict2','r','utf-8') as f:
        lines=f.readlines()
    dictJAEN={}
    kanji=''
    gloss=[]
    counter=0
    for line in lines:
        counter+=1

        jap_word_list=[]
        jap_word_list=''.join(line.split(' ')[0]).split(';')

        # get translations
        translations=[]
        for idx, item in enumerate(line.split('/')):
            if idx!=0 and re.search('\w+',item) and 'EntL' not in \
   item:
                #removing {} and anything in between
                item=re.sub('\{[^)]*\}','',item).strip()
                #removing ()
                item=re.sub('\([^)]*\)','',item).strip()
                #removing ) -because of nested paranthesis mess in\
    edict2-
                item=re.sub('\)','',item).strip()
                if not re.search('![ - ~]',item) and not re.search\
    ('\?',item) and item!='':
                    translations.append(item)
        for item in jap_word_list:
            item=re.sub('\([^)]*\)','',item).strip() #removing ()
            dictJAEN[item]=translations
    with codecs.open('resources/edict2.json','w','utf-8') as f:
        json.dump(dictJAEN,f)
```

A.3 Sentence Alignment Stage

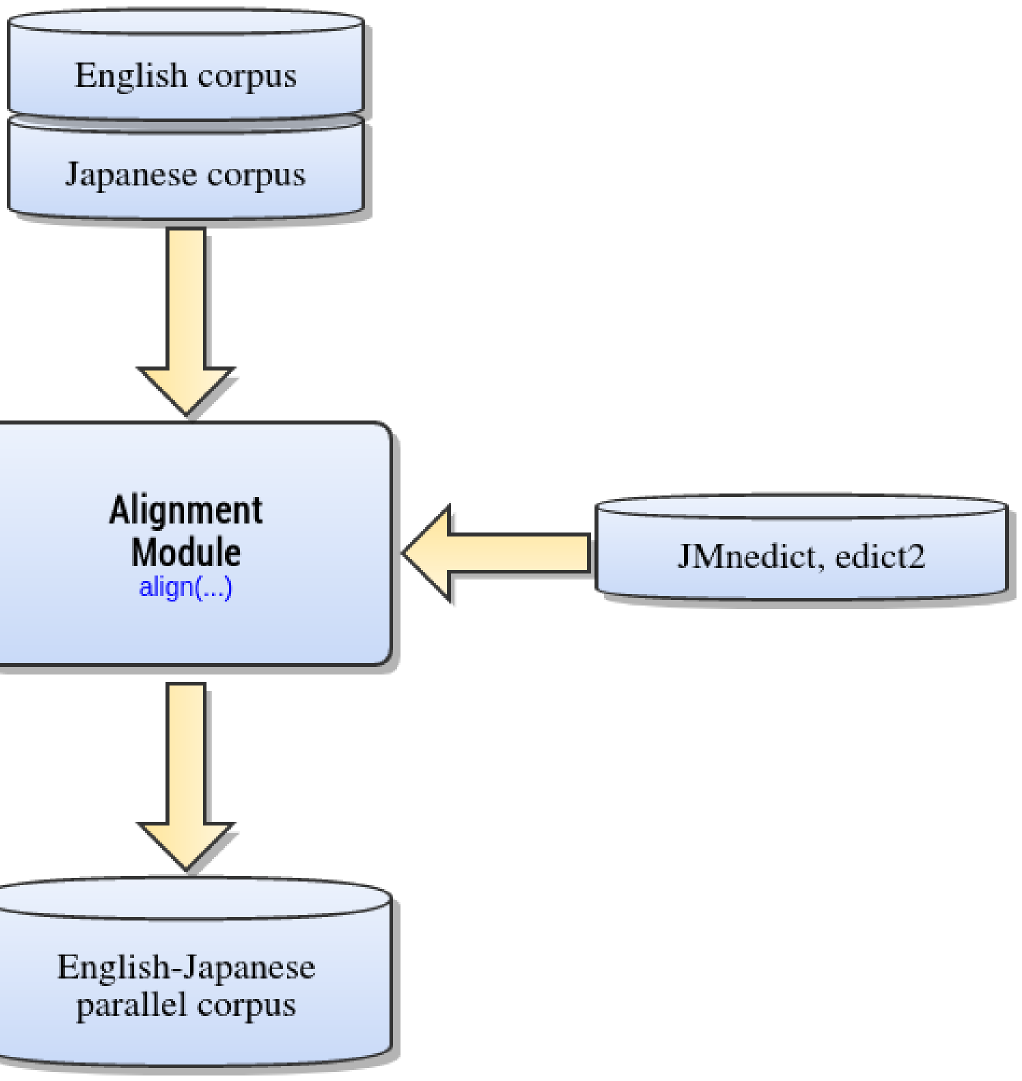


Figure A.3: Overview of the Sentence Alignment Stage. Module names are written in bold face, function names (without parameters) are shown in small, blue font.

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs
import json
import re
```

A.3.1 Alignment Module

Function – align

```python
def align():
    #dictionary files
    #**************************************************
    with codecs.open('JMnedict.json','r','utf-8') as f:
        JMnedict= json.load(f)
    with codecs.open('JMdict_e.json','r','utf-8') as f:
        JMdict= json.load(f)
    with codecs.open('edict2.json','r','utf-8') as f:
        edict2= json.load(f)
    #**************************************************

    #input files
    #**************************************************
    with codecs.open('japanese_pos.txt','r','utf-8') as f:
        pos_tags=f.readlines()
    with codecs.open('japanese_sentences_clean.txt','r','utf-8') \
   as f:
        text_jap=f.readlines()
    with codecs.open('english_sentences_clean.txt','r','utf-8') as\
    f:
        text_eng=f.readlines()
```

```
with codecs.open('english_sentences_clean_lemmatized.txt','r',\
'utf-8') as f:
    eng_lemmas=f.readlines()
#**************************************************

#output files
#**************************************************
f_out=codecs.open('sentence_align.txt','r+a','utf-8')
f_parallel=codecs.open('parallel.txt','a+','utf-8')
#**************************************************

temp_lines = f_out.readlines()
jap_counter=len(temp_lines)
translations=[]
match_dict={}
content_matches_list=[]
temp_counter=0
array=[] # metric values will be stored here
for idx in enumerate(eng_lemmas):
    # init to 0 for number of english sentences
    array.append(float(0))

for pos_tag in pos_tags:
    if 'EOS' in pos_tag:
        # weighted score algorithm
        #******************************************
        # find highest value in <array> array, index is the \
sentence number
        f_out.write('<ALIGN><JAP>'+str(jap_counter)+'<ENG>'+\
str(array.index(max(array))+1)+'<SCORE>'+str(max(array))+'\n')
        f_parallel.write(str(max(array))+' ||| '+text_jap[\
jap_counter-1].strip()+' ||| '+text_eng[array.index(max(array))\
].strip()+'\n')
        #******************************************
        jap_counter+=1 # japanese sentence counter
```

```
            for idx,val in enumerate(array):
                array[idx]=float(0)
            print 'Processing sentence '+str(jap_counter)+' of '+\
str(len(text_jap))
    jap_word=pos_tag.split('\t')[0]
    digit_matches=[]

    # do that only once per sentence (NB we're iterating POS \
tags!)
    if temp_counter!=jap_counter:
        temp_counter=jap_counter
        try:
            # finding sequences of numerals
            digit_matches=re.findall(r'\d+',text_jap[\
jap_counter-1])
        except:
            pass
        # if numerals were found, looking for equivalents in \
english data
        for match in digit_matches:
            jap_word=match
            eng_counter=0
            for eng_lemma in eng_lemmas:
                eng_counter+=1
                try:
                    if re.search(r'\b'+jap_word+r'\b',\
eng_lemma):
                        array[eng_counter-1]+=float(0.5)+float\
(float(1)/len(eng_lemma.split(' ')))
                except re.error,e:
                    print 'passing re.error at translation:',e
                    pass

    # look for romaji in japanese sentences (works badly with
    # numbers, since mecab tagging splits them up in single \
digits
     if '名詞'.decode('utf-8') in pos_tag and not re.search(r'\\
d+',jap_word):
       if re.search('\w',jap_word):
           eng_counter=0
           for eng_lemma in eng_lemmas:
               eng_counter+=1
               try:
                   if re.search(r'\b'+jap_word + r'\b', \
eng_lemma):
                       array[eng_counter-1]+=float(0.5)+float(\
float(1)/len(eng_lemma.split(' ')))
               except re.error,e:
                   print 'passing re.error at translation:',e
                   pass

     translations=[]
     # if named entity, look for match in nedict
     if '固有名詞'.decode('utf-8') in pos_tag:
         try:
             translations=JMnedict[jap_word]
         except KeyError,e:
             pass
     else: # else look for match in regular dictionary
         try:
             translations=JMdict[jap_word]
         except KeyError,e:
             pass
     # excluding certain japanese pos tags from dictionary \
lookup
     #(no particle, aux verb, etc)
     if not translations  and '助詞'.decode('utf-8') not in \
pos_tag and '助動詞'.decode('utf-8') not in pos_tag:
         try:
             translations=edict2[jap_word]
```

```
            except KeyError,e:
                pass

        if translations: # if any translation was found
            for translation in translations:
                eng_counter=0
                translation=re.sub('\(.*?\)','',translation)
                for eng_lemma in eng_lemmas:
                    eng_counter+=1
                    try:
                        if re.search(r'\b'+translation+r'\b',\
   eng_lemma):
                            array[eng_counter-1]+=float(0.5)+float\
   (float(1)/len(eng_lemma.split(' ')))
                            english_sentence_length=len(eng_lemma)
                            japanese_sentence_length=len(text_jap[\
   jap_counter-1].encode('utf-8'))
                    except re.error,e:
                        print 'passing re.error at translation: ',\
    e, translation
                        pass

        match_dict[jap_counter]=content_matches_list

    f_out.close()
    f_parallel.close()
```

B

Abstract

Abstract

The significance of sentence-aligned bilingual corpora, so-called parallel corpora, as training sets for machine translation systems and for various other language technology applications has become more and more evident in recent years. Even more desirable are collections which address a certain domain and hence offer more precise data for training of deep learning, statistical, or example-based approaches. The goal of this doctoral dissertation is to examine the feasibility of automated bilingual corpus creation from Wikipedia, specifically for languages which differ significantly in surface characteristics and other aspects. More precisely, how can Wikipedia be crawled to obtain domain-specific corpora in an efficient way, how can these corpora be sentence-aligned, and how can these alignments be evaluated to obtain the highest possible probability of a translated or equivalent sentence.

The research questions addressed in this work are: How much of the text on Wikipedia content can be used to build a bilingual aligned corpus for a specific language pair, and how can these texts be selected and aligned efficiently, all with minimal human input in the process.

The question is addressed by selecting two languages, which are representative of a dissimilar pair, English and Japanese. The resulting procedure, algorithms, software modules, and created corpus are a proof of concept, which can be adjusted in order to be applied to other dissimilar language pairs.

This dissertation proposes a method for crawling from Wikipedia by topic, aligning this data into a parallel corpus and a novel metric that measures the relative quality of this alignment. The resulting program tool chain is presented as a generic algorithm and is implemented in the Python programming language. The result of a first iteration of the software resulted in an English-Japanese parallel corpus of 66,000 sentence pairs. Human expert evaluations are presented to show the yield, feasibility, and efficiency of this method.

Abstract auf Deutsch

Die Wichtigkeit satz-alignierter bilingualer Korpora, auch paralle Korpora genannt, als Trainingsdaten für maschinelle Übersetzungsysteme und für eine Vielzahl anderer Sprachtechnologieanwendungen ist in den letzten Jahren immer deutlicher geworden. Sogar noch mehr gefragt sind Korpora, die eine bestimmte Domäne abdecken und somit noch zielgerichteter für das Training von Deep Learning, statistischen oder beispielbasierten Systemen sind. Das Ziel dieser Doktorarbeit ist es, die Realisierbarkeit der automatisierten Erstellung von parallelen Daten aus Wikipedia zu untersuchen. Insbesondere werden Sprachpaare untersucht, die in Hinblick auf Oberflächenstruktur und andere Aspekte sehr unterschiedlich sind. Genauer gesagt, wie kann domänenspezifischer Text aus Wikipedia effizient gesammelt werden, wie können diese Daten auf Satzebene aligniert werden und wie können diese Satzpaare evaluiert werden, um die bestmöglichen Übersetzungskandidaten zu bekommen.

Die Forschungsfragen sind: Wie viel des Wikipedia-Inhaltes kann verwendet werden, um bilinguale Korpora für ein bestimmtes Sprachpaar zu bauen und wie können diese Texte effizient aligniert werden; all das mit minimalem menschlichem Input.

Für die Beantwortung dieser Frage wurden zwei Sprachen gewählt, die repräsentativ für die Fragestellung sind, nämlich Englisch und Japanisch. Der Ablauf, die Algorithmen, die Softwaremodule und das daraus resultierende Korpus sind als Proof of Concept zu verstehen und können an andere Domänen und Sprachpaare angepasst werden.

Diese Arbeit schlägt eine Methode für themenspezifisches Datensammeln aus Wikipedia, eine Alignierungsmethode und eine Qualitätsmetrik vor. Die Algorithmen der in dem Zusammenhang entstandenen Software sind sowohl generisch beschrieben, wie auch in Python implementiert. Das Ergebnis einer Iteration der Software, 66,000 Satzpaare, ist der erste experimentelle Datensatz. Dieser Datensatz wird von Experten evaluiert, um die Ergiebigkeit, Umsetzbarkeit und Effizienz dieser Methode zu untersuchen.