Contributions for building a Corpora-Flow system

André Fernandes dos Santos
(andrefs@cpan.org)

Dissertation submitted in partial fulfillment of the requirements for the degree of Master in Informatics Engineering at the University of Minho, under the supervision of
José João Dias de Almeida
Anália Maria Garcia Lourenço

Departamento de Informática
Escola de Engenharia
Universidade do Minho

Braga, October 2011

Abstract

Text corpora are important resources on natural language processing and related areas such as biomedical text mining, corpus linguistics, machine learning and information extraction.

Preparing documents to be included in a corpus involves several different steps and a complex network of dependencies and conditions, which results in a workflow difficult to manage manually.

This dissertation focuses on different challenges which can be found when building corpora, and proposed methods to overcome such questions.

The first problem tackled was the cleaning of text documents – removing structural residues, normalizing encodings and notations and finding section delimiters – to make the documents suitable for further processing.

Another question addressed was the detection of duplicated documents and candidate document pairs for alignment. A method for measuring the similarity between documents was introduced and implemented.

Resumo

Os corpora textuais são um recurso importante no processamento de linguagem natural e em áreas relacionadas tais como a mineração de textos biomédicos, a linguística de corpus, aprendizagem máquina e recuperação de informação.

A preparação de documentos para inclusão num corpus envolve vários passos distintos e uma rede complexa de dependências e condições, que resulta num fluxo difícil de gerir manualmente.

Esta dissertação foca-se nos diversos desafios encontrados no processo de construção de corpora, e propõe métodos para ultrapassar essas questões.

O primeiro problema abordado foi a limpeza de documentos de texto – remoção de resíduos estruturais, normalização de formatos e notações e detecção de delimitadores de secção – tornando os documentos passíveis de serem processados.

Outra questão abordada foi a detecção de documentos duplicados e de pares de documentos candidatos a alinhamento, tendo sido introduzido e implementado um método para medição da similaridade entre documentos.

Then, the concept of document synchronization was introduced, followed by the description of an implementation based on section delimiters.

Posteriormente, introduziu-se o conceito de sincronização de documentos, seguido da descrição de uma implementação baseada nos delimitadores de secção.

Two real-world scenarios were used to guide the implementation of the tools developed: multi-language document alignment for inclusion in parallel aligned corpora and building corpora of biomedical texts for text mining.

Dois casos de estudo reais foram utilizados para guiar a implementação das ferramentas desenvolvidas: alinhamento multi-língua de documentos para inclusão em corpora paralelos alinhados e a construção de corpora de textos biomédicos para mineração de texto.

*A prototype of a corpora building management system was developed – a **corpora-flow system**. This system includes mechanisms which facilitate the implementation of the workflow needed to build a corpus.*

*Um protótipo de um sistema de gestão da construção de corpora foi desenvolvido – um **sistema de corpora-flow**. Este sistema incorpora mecanismos que facilitam a implementação do fluxo necessário para a construção de um corpus.*

A comparative evaluation of the set of tools developed was performed by aligning documents with and without using the tools developed. A small set of auxiliary tools was created to evaluate the results of alignments.

Uma avaliação comparativa do conjunto de ferramentas desenvolvido foi realizada através do alinhamento de documentos com e sem a intervenção das ferramentas desenvolvidas. Um pequeno conjunto de ferramentas foi desenvolvido para avaliar os resultados de alinhamentos.

Acknowledgments

- Many thanks to my teachers and supervisors, José João Almeida and Anália Lourenço, for all the time spent, the guidance and the ideas.
- Thanks again to Anália for the proof-reading, and for acting as my guide in the brave new world of bioinformatics and biomedical text mining.
- Thanks to Sara, Nuno and Alberto for all the help, the good advices and the discussions.
- Thanks again to Alberto, Nuno and José João for the borrowed templates and the tools and modules they made available.
- Thanks to all the friends from CeSIUM, for the friendship, the good moments and for letting me part of something great.
- Thanks to everyone at GroupBuddies, for sharing with me this adventure which is starting a brand new company.
- Thanks to Roberto, for all the crazy projects we've been through together – keep them coming!
- Thanks to everyone from Project Per-Fide, which helped me in my incursions in areas which I knew little about.
- Thanks to Fundação para a Ciência e Tecnologia, for granting me with a scholarship which made most of this work possible
- Thanks to my parents and my brother, for always having supported me and for being in great deal responsible for who I am today.
- And a special thanks to Dominique. Thank you for your infinite patience, for all the friendship, care, and for making me feel taller for the last 8 years! :)

Preface

This document is a master thesis in Informatics Engineering (area of Natural Language Processing) submitted to Universidade do Minho, Braga, Portugal.

Throughout the document the academic plural appears often in the text to describe the work developed. This form was intentionally used for two reasons: first, some of the work here presented was done in cooperation; and second, the plural can help the reader to feel more closely connected with the work done.

Document structure

Chapter 1 introduces the subject, defining the basic concepts and ideas developed throughout the document.

Chapter 2 presents some background on the concepts on which this dissertation is based, along with a brief overview of the state-of-the-art concerning these subjects.

Chapter 3 describes common types of noise found in text, the problem of cleaning plain text documents and the solution developed.

Chapter 4 analyzes the problem of measuring similarity between two documents, and how it can be used to detect duplicated documents and candidate pairs for alignment.

Chapter 5 presents the problem of unmatched sections in the alignment of documents, introduces the concept of book synchronization, and describes how it can be used to minimize the problem.

Chapter 6 introduces the concept of corpora-flow system, describes an implementation of a prototype for such a system based on Makefiles, and provides as an example the integration of the other developed tools.

Chapter 7 provides a global evaluation of the tools developed in the context of this project, based on tests performed with real documents.

Chapter 8 presents some conclusions and points out possible directions for future work.

Some complementary information is presented on the appendixes:

Appendix A includes the documentation for some tools developed through this thesis (also available as man pages).

Contents

List of Figures

List of Tables

List of Examples

Acronyms

API	application programming interface
ASCII	American Standard Code for Information Interchange
BTM	biomedical text mining
BoW	bag-of-word
CQP	Corpus Query Processor
DSL	domain-specific language
HMM	hidden Markov model
HTML	HyperText Markup Language
JSC	Jaccard similarity coefficient
LIE	language independent element
MWU	multi-word unit
NLP	natural language processing
OCR	optical character recognition
PDF	Portable Document Format
PTD	probabilistic translation dictionary
SL	source language
TL	target language
TM	translation memory
TMX	Translation Memory eXchange

TU translation unit

UCS Universal Character Set

UTF-8 UCS Transformation Format — 8-bit

Chapter 1

Introduction

The White Rabbit put on his spectacles. 'Where shall I begin, please your Majesty?' he asked.
'Begin at the beginning,' the King said gravely, 'and go on till you come to the end: then stop.'

Lewis Carroll, Alice's Adventures in Wonderland

1.1 Context and Motivation

Corpora are an invaluable resource for tasks in areas such as knowledge retrieval, machine learning, natural language processing and linguistics [Véronis, 2000]. Some corpora have a very well defined scope (e.g. legislation [Koehn, 2005], journalistic texts [Santos and Rocha, 2001, Santos and Sarmento, 2003] and biomedical [Kim et al., 2003, Stearns et al., 2001, Vincze et al., 2008]), while others have a wider focus. Either way, when building a corpus, usually it is desirable to obtain as many different documents from as many different sources as possible. The quantity of documents is important because most of the tools which work with corpora are based on statistics, and thus, corpora must be large enough to reduce the statistical bias. This means that the resources used to build them need to be available in reasonable amounts. The diversity of the documents is also important because the more different types and sources of documents there are, the richer the corpus will get, which tends to increase the precision and recall of the tools which work based on it. The downside of having many different sources of documents is an increase on the range of problems that they might present, the number of different notations used, and the probability of getting documents in bad conditions.

A parallel aligned corpus is composed by texts and their translation to another language (known as *bitexts*). When obtaining already aligned corpora is not possible, they can be built through the alignment of texts. This alignment is generally performed at paragraph, sentence or word level. Text alignment, as many other text processing tasks, may be performed manually; however, these tend to be tedious and time consuming tasks, which makes them unsuitable to be performed on large amounts of documents. Thus, whenever possible, the alternative is to use automatic tools.

Automatic tools have their disadvantages too, the largest one being that they are more affected by misformatted or noisy texts, or, generally, texts not in the very best conditions. This susceptibility to noisy text often means that tasks performed automatically need to be manually prepared, supervised and corrected. Without these steps, after the alignment, one might end up rejecting too many cases and obtain a corpus smaller than desirable, or accepting bad cases and obtain a low-quality corpora.

Document types are often a common source of problems when automatically processing text. Most tools can only operate over plain text, meaning that the documents must be converted before being processed. The conversion to plain text format often introduces noise and leaves residues, and even documents which were originally created in plain text format use different notations for things like mathematical formulas, section division, page breaks, sentences, etc. Text alignment suffers from additional problems, such as partial or truncated documents – aligning two versions of a given document is most of the times unsuccessful when one of them is missing even a small part, let alone entire sections or chapters.

The next section presents the work developed in the context of this dissertation, stating some design goals which guided the project development and briefly presenting the tools created. Section 1.3 introduces the case studies which served as motivation to this work and Section 1.4 presents the structure of the document, including a summary of each chapter.

1.2 Project overview

The context presented before justifies the need for tools which can help on the process of building corpora. This document will present work developed mainly in the context of Project Per-Fide (ref. PTDC/CLE-LLI/108948/2008), a project which aims to build a large parallel corpora (see Section 1.3.1, page 5), which is part of the reason why this work is mainly focused on

handling books.

1.2.1 Design Goals

A set of design goals was defined to guide implementation efforts. Some of these are practical requirements for the tools, while others can be viewed more as philosophical guidelines.

- Despite having the preparation of books for alignment as central concern, tools should be developed as generic as possible, because many of the problems they tackle are common to other fields.
- Develop tools which are independent from each other, but which can work together and with previously existing applications, and that can be assembled into a pipeline.
- Make the tools publicly available. This includes implementing them following the real-world standards and community practices, and making them installable and usable by third-party people and projects. This requires an extra effort given the difference between developing academic tools and real-world, usable applications.
- Whenever possible, identify and solve the problems automatically, generating a report of what was found and done. If a given problem is not solvable automatically, correctly identifying it and asking for confirmation is usually a viable alternative. When even this is not possible, presenting a report helps the user to decide what to do.
- Follow the *Unix philosophy* [Raymond, 2003]:

  > *Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*
  >
  > Doug McIlroy

  This will allow the development of tools which can help with the most common problems, and designed in such a way which makes them easy to integrate with smaller tools focused on more specific problems.
- Whenever possible, reuse existing well-proven tools. This includes Unix utilities such as *diff* and *grep* [Hunt et al., 1976, Hume, 1988].

1.2.2 Developed tools

This section presents and briefly describes the tools developed in the context of this master thesis, which include:

- a cleaner for plain text books and scientific articles.
- a tool to find book pairs and duplicated books in a collection.
- a tool to synchronize pairs of books.
- a prototype for a management tool for the corpora building workflow.
- small tools to evaluate and compare the results of alignments.

The tools were developed according to what was most needed at the time. As such, in this document they will not be presented necessarily in the chronological order in which they were created, but instead in a way that helps getting a global view of how they can fit together.

Plain text book cleaner

A Perl module and a command line utility which implement cleaning tasks in books – including removing page breaks, delimiting sections, normalizing sentence notation and other similar tasks and, in addition, a prototype for a similar tool but targeted at cleaning tasks specific for scientific articles, such as the normalization of mathematical formulas.

Duplicated documents and candidate pairs detector

A command line utility to detect duplicate and near duplicate files and candidate pairs for alignment, supported by a Perl module which implements algorithms for measuring similarity between files.

Book synchronizer

An utility to perform book synchronization – structural alignment based on the books' sections, capable of generating visual representations of the synchronization results and creating files with anchor points for alignment.

Corpora building workflow control system

Prototype for a workflow control system which can be used to create Makefile-controlled workflows.

Alignment results evaluation tools

Tools which help in the process of assessing the quality of the Translation Memory eXchange (TMX) files resulting from document alignment and comparing several TMX files.

1.3 Case Studies

Given the applied nature of this work, real-world use case scenarios can be used to support the *design-implement-test-evaluate* cycle of the development, as follows:

Design: the case studies allow to better decide which tools were needed, and which features should be implemented in those tools;

Implementation: many implementation decisions and optimizations could be triggered by the evaluation of the tool's outputs for the use cases;

Tests: the validation tests should be based on elements extracted from the case studies, eliminating the need of constructing artificial (often incomplete or biased) tests from scratch;

Evaluation: the evaluation of the tools should be performed against the case studies, asking simple questions: are these tools being useful in these real-world situations? What problems are not solved? What could be done to overcome/minimize them?

1.3.1 Multi-language document alignment

Project Per-Fide (ref. PTDC/CLE-LLI/108948/2008) is a project which aims to build and make publicly available a large annotated parallel corpora, containing texts in Portuguese in parallel with Spanish, Russian, French, Italian, German and English (Pt, Es, Ru, Fr, It, De, En – Per-Fide) [Araújo et al., 2010][1]. The project aims to have original texts in one of the seven languages

[1] `http://per-fide.di.uminho.pt`

and their translations, having Portuguese as either the source or target language.

The corpus is divided in two main genres: contemporary fiction (including, but not limited to, novels and short stories), and non-fiction: religious texts, journalistic articles, judicial texts, technical texts, and others.

Given the dimension of the tasks which it comprises, Project Per-Fide was not only the source of requirements for the tools but also has provided source documents to test and validate the tools developed.

1.3.2 Biomedical text mining

Areas of study within Life Sciences are particularly prolific in publishing scientific papers. In fact, the advances made in the last decades have originated not only a larger number of experiments, but also more complex experiments envisioning deeper levels of analysis.

The biomedical text mining community, comprised by specialists in areas such as natural language processing, machine learning and statistics, has endorsed multiple efforts trying to develop tools suited to process this kind of literature, and thus trying to fulfill the needs of the biologists [Rassinoux et al., 2010].

Despite featuring many specific tasks, at the basis of biomedical text mining the construction of corpora. As such, biomedical text mining appeared as a second natural case study, giving us the possibility to develop tools with a wider range of application, allowing us to apply the techniques and algorithms developed in different contexts, and preventing us from over-specializing our tools.

1.4 Document Summary

Background

The next chapter presents some background information and the state of the art in the area of corpora construction, text alignment (given that this is one of the ways of producing parallel corpora) and text handling.

Cleaning documents

This chapter discusses the problem of processing noisy text, and describes the most common types of noise. Several methods to perform cleaning tasks

are proposed, including removing structural residues, normalizing encodings and notations, and finding sections delimiters.

Measuring similarity

This chapter introduces the problem of finding duplicated documents and finding candidate pairs of documents for alignment. A solution to this problem is discussed, based on a method for measuring the similarity between documents by extracting bags-of-words (BoWs) from the text.

Synchronizing books

In this chapter the problem of aligning documents with missing sections is presented. A method to solve it is proposed, based on the identification of the missing sections by synchronizing the documents (aligning them at section level). The section delimiting is performed by the method described in **Cleaning documents**.

Prototype of a corpora flow

This chapter describes a prototype for a system to generate corpora building workflow managers, through the extension of the concept of Makefiles.

Global evaluation

This chapter describes a global evaluation of the tools developed in the context of this dissertation. A small set of tools was created to evaluate alignment results, and a comparative analysis of alignments with and without the tools was performed.

Conclusions and future work

This chapter presents a review over the work performed, the problems tackled and the tools implemented. Several possible future work directions are described.

Chapter 2

Background

He wrote it in simple declarative sentences with all of the problems ahead to be lived through and made to come alive. The very beginning was written and all he had to do was go on.

Ernest Hemingway, Garden of Eden

2.1 Introduction to corpora

This section introduces the concept of corpus, explaining what it is and what types of corpora exist.

2.1.1 What is a corpus?

In the broader sense of the word, a corpus is any finite collection of multiple documents. Depending on the context, the notion of what a corpus is may change a little, and it has even been shifting over time (specially with the widespread generalization of the Internet), which means that does not exist a unique definition of what a corpus is.[1] Sinclair [2005] defines corpus as follows:

[1] Some authors use the Oxford Dictionary definition of corpus:
Corpus: *a collection of written texts, especially the entire works of a particular author or a body of writing on a particular subject; collection of written or spoken material in machine-readable form, assembled for the purpose of linguistic research.* [Dictionaries, April 2010]

Definition 1 *A **corpus** is a collection of pieces of language text in electronic form, selected according to external criteria to represent, as far as possible, a language or language variety as a source of data for linguistic research.* ◇

Areas such as biomedical text mining (BTM) have their own notion of corpus, which is generally a large collection of documents, usually annotated and used to train, test or validate an automatic tool.

2.1.2 Types of corpora

Corpora are usually divided into text corpora (i.e. written material), spoken corpora and multimodal corpora. Below are presented some common types of specialized text corpora. A comprehensive list of corpora was compiled by Xiao [2008].

Treebanks: linguistically annotated corpus that includes some grammatical analysis beyond the part-of-speech level;

Historical corpus: one which is intentionally created to represent and investigate past stages of a language and/or to study language change;

Learner corpus: A corpus where all the data comes from language learners.

Parallel and comparable corpus: multilingual corpus which is a) composed by texts and their translation (parallel corpora); or b) composed by texts in different languages which are not translations but are somehow similar to each other – according to previously defined parameters.

Corpora can be further classified according to internal characteristics. They may be general corpus or specific corpus, in which case they may vary in the genre of texts included, the topics covered, the size and the annotations and the languages covered.

Texts genre: Corpora may be focused on a particular genre of text. For example, the Europarl Parallel Corpus contains European legislation and CETEMPublico and CETEM-Folha were put together using journalistic texts [Santos and Rocha, 2001, Santos and Sarmento, 2003].

Topics covered: Some corpora are specialized in a given topic or area of knowledge. For example, the GENIA corpus comprised abstracts from biomedical articles on human transcriptional regulation [Kim et al., 2003].

Corpus size: The (minimum) size of a corpus should depend on two main criteria: the kind of query that is anticipated from users, and the methodology they use to study the data [Sinclair, 2005]. The size is usually measured in total number of words and total number of different words.

Annotations: Many kinds of annotations can be added to a corpus: part-of-speech, lemmatisation, parsing, semantics, discoursal and text linguistic annotations, etc. A comprehensive and detailed description of common types of annotations can be found in McEnery and Wilson [2001].

Languages covered: Corpus may be monolingual, in which case all the texts are written in the same language, or multilingual (see above the description of parallel and comparable corpora).

2.2 Building corpora

The steps required to build a corpus may depend partially on the size of the corpus, its purpose, and its scope – whether it is a specific corpus (one which only includes a given genre of text or that only covers a specific topic) or a general corpus. Nevertheless, it is possible to define a list of tasks commonly associated to the process of building a corpus.

2.2.1 Common tasks

The following list has been compiled from the existing literature on building corpora [Wynne et al., 2005, Burnard, 1999, Atkins et al., 1992], and includes the most common and important tasks required to constitute a corpus.

Planning

Like any other project, building a corpus typically starts with the planning phase, where several decisions need to be made which will affect the remaining stages. Some of these decisions, however, will most likely have to be revised and adapted during the building process when faced with practical aspects of the project. Many of these details will also be highly dependent on constraints such as the time and funds available.

The type and amount of texts that will be included in the corpus must be defined. Concerns such as the corpus being balanced and providing a good coverage (for example, in terms of textual genres and topics included) should

be taken into account. Selecting the documents that should be included will also depend on the intended use for the corpus and their availability. The procedures needed for inclusion of documents should also be established: where and how to obtain documents, and how to get the rights clearance and permissions to include the documents in the corpus.

Defining the intermediary file formats and conventions to be used in the text processing pipeline allows everybody involved in the project to create documents which can be easily understood by the tools developed, avoiding the need of reformatting or converting files. These details should also be defined at the planning phase.

Then there are the aspects related to the use of corpora: what annotations should be added to the corpus and how should the corpus be made available – in which formats should it be available and how are the analysis tools are supposed to interact with it.

Getting permissions

The documents which are in the public domain usually do not require any special permission, as they are free for anyone to use. Copyrighted documents, on the other hand, can only be used in corpora if their copyright owners explicitly allow it.

Clearing the rights to use documents is a time-consuming task, which involves identifying corporations or other entities which are more likely to detain a relevant set of documents and contact them, asking for the permissions to use their documents in the project and waiting for their reply. Often, getting a reply requires several contact attempts and climbing up the institutions' hierarchy. When the permission is granted, it is also common for the copyright owner to impose some specific constraints – e.g. only allowing to sample a given percentage of non-consecutive sentences – or to demand some counterparts – e.g. requiring some specific type of annotation or access to the final version of the corpus.

Gathering documents

Copyright issues to the side, documents must be retrieved. Nowadays, there are three main ways of putting together a text corpus: obtaining already digitized texts, scanning printed documents, or typing in existing material.

Documents in electronic format are usually the ones that require less efforts to make suitable for the corpus, but a fair amount of parsing and refor-

matting is often still required.

It is not always possible to rely solely on documents available in electronic formats. This may happen due to the nature of the corpus – documents in less popular languages, genres or topics may be harder to find in large quantities, and historical documents are less likely to be digitized. In these cases, the best alternative is usually to scan them and perform the optical character recognition (OCR). Even so, the resulting files have to be manually checked for errors, which makes this a time and effort consuming task.

Sometimes, not even the OCR is possible. This is the case with documents which are too complex or degraded, handwritten documents and whenever the transcription of audio recordings is required. In these cases, manually typing in the text is the only alternative. However, this is impractical (even impossible) to do in large quantities. This means that relying too much on manually typed documents limits the size of the corpus and delays the project, reducing its importance.

2.2.2 Parallel corpora

Building parallel corpora presents some specificities because it requires the fetching of parallel documents – documents containing the same texts in different languages. Before being included in the corpus, the bitexts must be aligned, which requires the documents to be fetched already aligned or the alignment to be performed after getting the documents.

The process of alignment is described in detail in Section 2.3.3. Below is compiled a list of common sources for parallel documents.

Document sources

As mentioned in section 2.3.1, the lack of available corpora was one of the reasons appointed to why the earlier efforts made to align texts did not work out. Fortunately, with the massification of computers and globalization of the internet, several sources of alignable corpora have appeared:

Literary Texts

Books published in electronic format are becoming increasingly common. Despite the vast majority being subject to copyright restrictions, some of them are not.

Project Gutenberg, for example, is a volunteer effort to digitize and archive cultural works. Currently it comprises more than 33.000 books,

most of them in public domain [Hart and Newby, 1997]. National libraries are beginning to maintain a repository of eBooks as well [Varga et al., 2005].

Sometimes it is also possible to find publishers who are willing to cooperate by providing their books as long as it is for research purposes only.

Religious Texts

The Bible has been translated into over 400 languages, and other religious books are widespread as well. Additionally, the Catholic Church translates papal edicts to other languages from the original Latin, and the Taizé Community website[2] also provides a great deal of its content in several languages.

International Law

Important legal documents, such as the Universal Declaration of Human Rights[3] or the Kyoto Protocol[4] are freely available in many different languages. One of the first corpora used in parallel text alignment were the Hansards – proceedings of the Canadian Parliament [Brown et al., 1991, Gale and Church, 1993, Kay, 1991, Chen, 1993].

Movie subtitles

There are several on-line databases of movie subtitles. Depending on the movie, it is possible to find dozens of subtitles files, in several different languages (frequently even more than one version for each language). Subtitles are shared on the internet as plain text files (sometimes tagged with extra information such as language, genre, release year, etc) [Tiedemann, 2007].

Software internationalization

There is an increasing amount of multilingual documentation belonging to software available in several countries. Open source software is particularly useful for not being subject to copyright [Tiedemann et al., 2004].

Bilingual Magazines

Magazines like National Geographic, Rolling Stone or frequent flyer magazines are often published in other languages besides English, and in

[2] `http://www.taize.fr`

[3] `http://www.un.org/en/documents/udhr/index.shtml`

[4] `http://unfccc.int/kyoto_protocol/items/2830.php`

several countries there are magazines with complete mirror translations into English.

Websites

Websites often allow the user to choose the language in which they are presented. This means that a web crawler may be pointed to this websites to retrieve all reachable pages – a process called "mining the Web" [Resnik and Smith, 2003, Tsvetkov and Wintner, 2010].

Corporate webpages are one example of websites available in several languages. Additionally, some international companies have their subsidiaries publishing their reports in both their native language and in their common language (usually English).

2.3 Text alignment

This section describes the evolution of text alignment since its beginnings and relevant alignment and evaluation projects. Then, a detailed step-by-step description of the alignment process is made, and the section ends with a list of current projects and tools.

2.3.1 Brief history

Parallel text use in automatic language processing was first tried in the late fifties, but probably due to limitations in the computers of this time (both in storage space and computing power) and the reduced availability of large amount of textual data in a digital format, there was a restricted use of corpora at time and the results were not encouraging [Véronis, 2000].

In the eighties, with computers being several orders of magnitude faster, a similar increase in storage capacity and a growing amount of large corpora available, a new interest in text alignment emerged, resulting in several publications years later.

The first documented approaches to sentence alignment were based on measuring the length of the sentences in each text. The system developed by Kay [1991] was the first to devise an automatic parallel alignment method. This method was based on the idea that when a sentence corresponds to another, the words in them must also correspond. All the necessary information, including lexical mapping, was derived from the texts themselves.

The algorithm initially sets up a matrix of correspondence, based on sentences which are reasonable candidates: the initial and final sentences have

a good probability to correspond to each other, and the remaining sentences should be distributed close to the matrix main diagonal. Then the word distributions are calculated, and words with similar distribution values are matched, and considered anchor points, narrowing the "alignment corridor" of the candidate sentences. The algorithm then iterates until it converges on a minimal solution.

However, this system was not efficient enough to apply to large corpora [Moore, 2002]. Two other similar methods were proposed, the main difference between them being on how the sentence length was measured: Brown et al. [1991] counted words, while Gale and Church [1993] counted characters. The authors assumed that the length of a sentence is highly correlated with the length of its translation. Additionally, they concluded that there is a relatively fixed ratio between the sentence length in any two languages. The optimal alignment minimizes the total dissimilarity of all the aligned sentences. Gale and Church [1993] achieve the optimal alignment with a dynamic programming algorithm, while Brown et al. [1991] applied hidden Markov models (HMMs).

In 1993, Chen [1993] proposed an alignment method which relied on additional lexical information. In spite of not being the first of its kind, this algorithm was the first lexical-based alignment which was efficient enough for large corpora. It required minimum human intervention – at least 100 sentences had to be aligned for each language pair to bootstrap the translation model – and it was capable of handling large deletions in text.

Another family of algorithms was proposed soon after, based on the concept of cognates. The concept of cognate may vary a little, but generally cognates are defined as occurrences of tokens that are graphically or otherwise identical in some way. These tokens may be dates, proper nouns, some punctuation marks or even closely-spelled words – Simard and Plamondon [1998] considered cognates any two words which shared the first four characters. When recognizable elements like dates, technical terms or markup are present in considerable amounts, this method works well even with unrelated languages, despite some loss of performance being noticeable.

Most methods developed ever since rely on one or more of these main ideas – length based, lexical or dictionary based, or partial similarity (cognate) based.

2.3.2 Alignment and evaluation projects

ARCADE

In the mid-nineties, the amount of studies describing alignment techniques was already impressive. Comparing their performance, however, was difficult due to the aligners sensitivity to factors as the type of text used or different interpretations of what is a "correct alignment". Details on the protocols or the evaluation performed on the systems developed were also frequently not disclosed. A precise evaluation of the techniques would be most valuable; yet, there was no established framework for evaluation of parallel text alignment.

The ARCADE I was a project to evaluate parallel text alignment systems which took place between 1995 and 1999, consisting in a competition among systems at the international level [Langlais et al., 1998, Véronis and Langlais, 2000]. It was divided in two phases: the first two years were spent on corpus collection and preparation, and methodology definition, and a small competition on sentence alignment was held; in the remaining time, the sentence alignment competition was opened to a larger number of teams, and a second track was created to evaluate word-level alignment.

For the sentence alignment track, several types of texts were selected: institutional texts, technical manuals, scientific articles and literature. To build the reference alignment, an automatic aligner was used, followed by hand-verification by two different persons.

F-score values were based on the number of correct alignments, proposed alignments and reference alignments, and calculated for different types of granularity. The best systems achieved an F-score of over 0.985 on institutional and scientific texts. On the other hand, all systems performed poorly on the literature texts.

For the word track, the systems had to perform translation spotting, a sub-problem of full alignment: for a given word or expression, the objective is to find its translation in the target text. A set of sixty French words (20 adjectives, 20 nouns and 20 verbs) were selected based on frequency criteria and polysemy features. The results were better for adjectives (0.94 precision and recall for the best system) than for verbs (0.72 and 0.62 respectively). The overall results were 0.77 and 0.73 respectively for the best system.

The ARCADE project had a few limitations: the systems were tested by limited tasks which did not reflect their full capacity, and only one language pair (French-English) was tested. Nonetheless, it allowed to conclude, at the time, that the techniques were satisfactory for texts with a similar structure. The same techniques were not as good when it came to align

texts whose structure did not match perfectly. Methodological advances on sentence alignment and, in a limited form, word alignment were also made possible, resulting in methods and tools for the generation of reference data, and a set of measures for system performance assessment. Additionally, a large standardized bilingual corpus was constructed and made available as a gold standard for future evaluations.

ARCADE-II

The second campaign of ARCADE was held between 2003 and 2005, and differed from the first one in its multilingual context and on the type of alignment addressed. French was used as the pivot language for the study of 10 other language pairs: English, German, Italian and Spanish for western-European languages; Arabic, Chinese, Greek, Japanese, Persian and Russian for more distant languages using non-Latin scripts.

Two tasks were proposed: sentence alignment and word alignment. Each task involved the alignment of Western-European languages and distant languages as well. For the sentence alignment, a pre-segmented corpus and a raw corpus were provided.

The results showed that sentence alignment is more difficult on raw corpora (and also, curiously, that German is harder than the other languages). The systems achieved an F-score between 0.94 and 0.97 on raw corpora, and between 0.98 and 0.99 on the segmented one. As for distant languages alignment, two systems (P1 and P2) were evaluated. The results allowed to conclude that sentence segmentation is very hard on non-Latin scripts, as P1 was incapable of performing it and the results for raw corpora alignment of P2 scored 0.421.

Evaluating word alignment presents some additional difficulties, given the differences in word order, part-of-speech and syntactic structure, discontinuity of multi-token expressions, etc which are possible to find between texts and its translations. Sometimes it is not even clear how some words should be aligned when they do not have a direct correspondence in the other language. Nevertheless, a small competition was held, in which the systems were supposed to identify named entities phrases translation in the parallel text.

The scope of this task was limited; yet, it allowed to define a test protocol and metrics.

Blinker project

In 1998, a "bilingual linker", Blinker, was created in order to help bilingual annotators (people who go through bitexts and annotate the correspondences between sentences or words) in the process of linking word tokens that are mutual translations in parallel texts. This tool was developed in the context of a project whose goal was to manually produce a gold standard which could be used to future research and development [Melamed, 1998b]. Along with these tools, several annotation guidelines were defined [Melamed, 1998a], in order to increase consistency between the annotations produced by all the teams.

The parallel texts chosen to align were a modern French version and a modern English version of the Bible. This choice was motivated by the canonical division of the Bible in verses, which is common across most of the translations. This was used as a "ready-made, indisputable and fairly detailed bitext map".

Several methods were used to improve reliability. First, as many annotators as possible were recruited. This allowed to identify deviations from the norm, and to evaluate the gold standard produced in terms of inter-annotator agreement rates (how much the annotators agreed with each other).

Second, Blinker was developed with some unique features, such as forcing the annotator to annotate all the words in a given sentence before proceeding (either by linking them with the corresponding target text word, or by declaring as "not translated". These features aimed to ensure that a good first approximation was produced, and to discourage the annotators natural tendency to classify words whose translation was less obvious as "not translated".

Third, the annotation guidelines were created to reduce experimenter bias, and a monetary bonus was offered to the annotators who stayed closest to the guidelines.

The inter-annotator agreement rates on the gold standard were approximately 82% (92% if function words were ignored), which indicated that the gold standard is reasonably reliable and that the task is reasonably easy to replicate.

2.3.3 The alignment process

Despite the existence of several methods and tools to align corpora, the process itself shares some common steps. In this section the alignment process is detailed step-by-step.

Input

The format of the documents to be aligned usually depends on where they come from: while literary texts are often obtained either as PDF or plain text files; legal documents usually come as PDF files; corporate websites come in HTML; and movie subtitles are either in SubRip or microDVD format.

Globally accepted as the standard format to share text documents, PDF files are usually converted to some kind of plain text format: either unformatted plain text, HTML or XML. In spite of being available several tools to perform this conversion, given that the final layout of PDF files is dictated by graphical directives, without the notion of text structure, frequently the final result of the conversion is far from perfect: remains of page structure, loss of text formatting and noise introduction are common problems. A comparative study of the tools available to convert PDF to text can be found in Robinson [2001].

HTML or XML files are less problematic. Besides being more easily processed, some of the markup elements may be used to guide the alignment process: for example, the header tags in HTML may be used to delimit the different sections of the document, and the text formatting tags as `<i>` or `<b>` might be preserved and used as well. Depending on the schema used, XML annotations can be an useful source of information too.

Document alignment

Frequently, a large number of documents is retrieved from a given source with no information provided of which documents match each other. This is typically the case when documents are gathered from websites with a crawler.

Fortunately, when there are multiple versions available, in different languages, of the same page, it is common to include in the name a substring identifying the language: for instance, `Portuguese`, `por` or `pt` [Almeida and Simões, 2010]. These substrings usually follow either the ISO-639-2[5] or the ISO 3166[6] standards. This way, it is possible to write a script with a set of rules which help to find corresponding documents.

Other ways to match documents are to analyze their meta-information, or the path where they were stored.

[5] `http://www.loc.gov/standards/iso639-2/php/code_list.php`
[6] `http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm`

Paragraph/sentence boundary detection

The higher levels of alignment are section, paragraph or, most commonly, sentence alignment. In order to perform sentence-level alignment, texts must be segmented into sentences. Some word-level aligners work based on sentence-segmented corpora as well, which allows them to achieve better results.

Splitting a text into sentences is not a trivial task because the formal definition of what is a sentence is a problem that has eluded linguistic research for quite a while (see Simard [1998] for further details on this subject). Véronis and Langlais [2000] give an example of several valid yet divergent segmentations of sentences.

A simple heuristic for a sentence segmenter is to consider symbols like `.!?` as sentence terminators. This method may be improved by taking into account other terminator characters or abbreviations patterns; however, this patterns are typically language-dependent, which makes it impossible to have an exhaustive list for all languages[7] [Koehn, 2005].

This process usually outputs the given documents with the sentences clearly delimited.

Sentence Alignment

There are several documented algorithms and tools available to perform sentence-level alignment. Generally speaking, they can be divided into three categories: length-based, dictionary or lexicon based or partial similarity- based (see Section 2.3.1 on page 15 for more information).

Generally, sentence aligners take as input the texts to align, and, in some cases, additional information, such as dictionaries, to help establish the correspondences.

A typical sentence alignment algorithm starts by calculating alignment scores, trying to find the most reliable initial points of alignment – denominated "anchor points". This score may be calculated based on the similarity in terms of length, words, lexicon or even syntax- tree [Tiedemann, 2010]. After finding the anchor points, the process is repeated, trying to align the middle points. Typically, this ends when no new correspondences are found.

The alignment is performed without allowing cross-matching, meaning that the sentences in the source text must be matched in the same order in the target text.

[7] An example of a sentence segmenter can be downloaded at `http://www.eng.ritsumei.ac.jp/asao/resources/sentseg/`.

Several outcomes are possible for each correspondence found: the most common case is when a source text sentence corresponds exactly to a target text sentence (1:1). Less frequently, there are omissions (1:0), additions (0:1) or something more complex (m:n), usually with $1 \leq m, n \leq 2$. An example of a sentence-level alignment is presented in Table 2.1.

Table 2.1: Extract of sentence-level alignment performed using Portuguese and Russian subtitles from the movie Tron.

Portuguese	**Russian**
A actividade do laser começará em 30 segundos.	Лазер будет включен через 30 секунд.
Ponham os óculos de proteção. Abandonem a área.	Наденьте защитные очки и покиньте главный зал
Deixe-me ver se nós temos a luz verde.	Интересно, будет ли на этот раз зелёное свечение...
Área do alvo protegida?	- Облучаемая область свободна?
Verificaram a segurança.	- Да , её уже проверили.
20 segundos	20 секунд.
- Parece bom.	- Вроде всё нормально.
- Deixe-o iniciar.	- Можно запускать.
Isto é o que nós temos esperado.	Включаю.

Tokenizer

Splitting sentences into words allows to subsequently perform word-level alignment. As with sentence segmenting, it is possible to implement a very simple heuristic to accomplish this task by defining sets of characters which are to be considered either word characters or non-word characters. For example, in Portuguese, it is possible to define `A-Z`, `a-z`, `-` and `'` (and all the characters with diacritics) as belonging inside words, and all others as being non-word characters.

However, here too there are languages in which other methods must be used: in Chinese, for example, word boundaries are not as easy to determine as with Western-European languages. This imposes interesting challenges on how to align texts in some languages at word-level.

Word alignment

The word-alignment process presents some similarities with higher level alignments. However, this is a more complex process, given the more frequent

order inversions, differences in part-of-speech and syntactic structure, and multi-word units (MWUs). As a result, research in this area is less advanced than in sentence alignment [Véronis and Langlais, 2000, Chiao et al., 2006].

MWUs are idiomatic expressions composed by more than one word. MWUs must be taken into account because frequently they cannot be translated word-by-word; the corresponding expression must be found, whether it is also an MWU or a single word [Tiedemann, 1999, 2004]. This happens often when aligning English and German, for instance, because of the German composed words such as *Geschwindigkeitsbegrenzung* (in English, *speed limit*).

Output

The output formats vary greatly. Generally, every format must include some kind of sentence identification (either by the offsets of its first and last characters or by a given identification code) and the pairs of correspondences; optionally, additional information may also be included, like a confidence value, or the stemmed form of the word.

The BAF corpus, for example, is available in the following formats [Simard, 1998]:

COAL format: An alignment in this format is composed by three files: two plain text files containing the actual sentences and words originally provided, and an alignment file which contains a sequence of pairs $[(s1, t1), (s2, t2), ..., (sn, tn)]$. Each s_i^{th} segment in the source text has its beginning position given by s_i, and the end position given by $S_{i+1} - 1$. The corresponding segment in the target text is delimited by t_i and $t_i{+}1 - 1$.

CES format: This format is also composed by three files: two text files in CESANA format, enriched with SGML mark-up that uniquely identifies each sentence, and an alignment file in CESALIGN format, containing a list of pairs of sentence identifiers. The results of ARCADE II were also stored under this format.

HTML format: Not intended for further processing purposes. This is a visualization format, displayable in a common browser.

Other possible formats are translation memory (TM) formats [Désilets et al., 2008]. TMs are databases (in the broader sense of the word) that store pairs of "segments" (either paragraphs, sentences or words). TMs store the source text and its translation in language pairs denominated translation units. There are multiple TMs formats available.

2.3.4 Current projects and tools

This section lists some of the most relevant projects and tools being developed or used at the moment.

NATools

NATools is a set of tools for processing, analyze and extract translation resources from parallel corpora, developed at University of Minho. It includes sentence and word aligners, a probabilistic translation dictionary (PTD) extractor, a corpus server, a set of corpora and dictionary query tools and tools for extracting bilingual resources [Simões and Almeida, 2007, 2006, 2003].

Giza++

Giza++ is an extension of an older program, Giza. This tool performs statistical alignment, implementing several HMMs and advanced techniques which allow to improve alignment results [Och and Ney, 2000].

hunalign

hunalign is a sentence-level aligner built on top of the algorithm proposed by Gale and Church [1993], written in C++. When provided with a dictionary, hunalign uses its information to help in the alignment process, despite being able to work without one [Varga et al., 2005].

Per-Fide

Per-Fide is an undergoing project from University of Minho which aims to compile parallel corpora between Portuguese and six other languages (see Section 1.3.1 on page 5).

cwb-align

Also known as easy-align, this tool is integrated in the IMS CWB Open Corpus Workbench, a collection of open source tools for managing and query large text corpora with linguistic annotations, based of an efficient query processor, the Corpus Query Processor (CQP) [Evert, 2001]. It is considered as a standard *de facto*, given its quality and implemented features, such as tools for encoding, indexing, compression, decoding, and frequency distributions, a

query processor and a CWB/Perl API for post-processing, scripting and web interfaces.

WinAlign

WinAlign is a commercial solution from the Trados package, developed for professional translators. It accepts previous translations as translations memories and uses it to guide further alignments. This is also useful when clients provide reference material from previous jobs.

Chapter 3

Cleaning documents

What was the state of that scrap of paper when you found it? Was it clean or dirty?

Agatha Christie, Sad Cypress

This chapter addresses the problems caused by several types of noise found in texts which affect most of further processing, and describes the implementation of several methods to clean books and getting them ready for alignment. Additionally a similar tool is presented, devoted to the cleaning and preparation of scientific papers, making them suitable for text mining.

3.1 Introduction

3.1.1 Motivation

Many tasks within the natural language processing (NLP) field have text as their starting point, and the quality of results that can be obtained is tightly connected to the text quality itself.

Several types of noise may be present in texts, resulting in a wrong interpretation of individual characters (e.g. Greek letters, sub/superscripts, dashes and apostrophes); words, phrases and sentences; paragraph frontiers; tables; pages, headers, footers and footnotes; titles and section frontiers.

Although some of these problems are more likely to be found in some genres of text than others, their occurrence is usually more dependent on the type of the document, its origin and intended use.

3.1.2 Common problems

The main characteristics of documents which may negatively affect their processing (and consequently, their alignment [Véronis, 2000]) are:

File format: Documents available in PDF or Microsoft Word formats (or, generally, any structured or unstructured format other than plain text) need to be converted to plain text before being processed, using tools such as `pdftotext` [Noonburg, 2001] or Apache Tika [Gupta and Ahmed, 2007, Mattmann and Zitting, 2011]. This conversion often leads to loss of information (text structure, text formatting, images, etc) and the introduction of noise (bad conversion of mathematical expressions, diacritics, tables and other) [Robinson, 2001];

Text encoding format: There are available several text encoding formats. For example, Western-European languages can be encoded as ISO-8859-1 (also known as Latin1), Windows CP 1252, UTF-8 or others. Discovering the encoding format used in a given document is frequently not a straightforward task, and dealing with a text while assuming the wrong encoding may have a negative impact on the results;

Structural residues: Some texts, despite being in plain text format, still contain structural elements from their previous format (for example, it is common to find page numbers, headers and footers in the middle of the text of documents which have been converted from PDF to plain text);

Sectioning notation: There are endless ways to represent and numerate the several divisions and subdivisions of a document (parts, chapters, sections, etc). Several of these notations are language- dependent (e.g. Part One or *Première Part*). Being able to detect the section divisions allows to differentiate and process them differently.

Additionally, in the particular case of text alignment, identifying the section titles can be used to guide the alignment process by pairing the sections first (more information on this subject can be found on Chapter 5).

Unpaired sections: Sections present an additional problem in the case of text alignment: sometimes one of the documents to be aligned contains one or more sections which are not present in the other document. For example, forewords to a given edition, preambles, etc.

These problems are an obstacle in text processing, and are often enough to derail the alignment process, producing unacceptable results. As such, these questions are found frequently enough to justify the development of a tool to pre-process books, clean them and prepare them for further processing.

There are other tools available which also address the issue of preparing corpora for alignment, such as the one described by Okita [2009]; or text cleaning, such as TextSoap, a program for cleaning text in several formats [UnmarkedSoftware, 2011].

In order to solve the problems previously described, the first approach was to implement a Perl script which, with the help of UNIX utilities like `grep` and some regular expressions, attempted to find the undesired elements and normalize, replace or delete them. As we started to have a better grasp on the task, it became clear that such a naive approach was not enough.

A more sophisticated approach was then envisioned, resulting in the creation of `Text::Perfide::BookCleaner` (`T::P::BC`), a Perl module divided in separate functions, each one dealing with a specific problem.

3.1.3 Design Goals

`Text::Perfide::BookCleaner` was developed with several design goals in mind. Each component of the module meets the following three requirements:

Optional: depending on the conditions of the books to be cleaned and what they are going to be used for, some steps of the process may not be desired or necessary. Thus, each step may be performed independently from the others, or not be performed at all.

Chainable: the functions can be used in sequence, with the output of one function passing as input to the next, with no intermediary steps required.

Reversible: no information is lost (i.e. everything removed from the original document is kept in separate files). This means that, at any time, it is possible to revert the book to a previous (or even the original) state.

3.2 Cleaning books

We decided to tackle the problem in five different steps: *pages*, *sections*, *paragraphs*, *footnotes*, and *words and characters*. Each of these steps deals with a specific type of problems commonly found in plain text books. A

commit option is also available as a final step which irreversibly removes all the markup added along the cleaning process. The different steps are implemented as functions in `Text::Perfide::BookCleaner`. A Perl script, `bookcleaner`, implements the whole workflow.

Figure 3.1 presents the pipeline of the `Text::Perfide::BookCleaner` module. `bookcleaner` accepts as input documents in plain text format, having as default UTF-8 text encoding. The final output consists of a summary report and the cleaned version of the original document, which may then be used in several processes, such as alignment [Varga et al., 2005, Evert, 2001], ebook generation [Lee et al., 2002], or others.

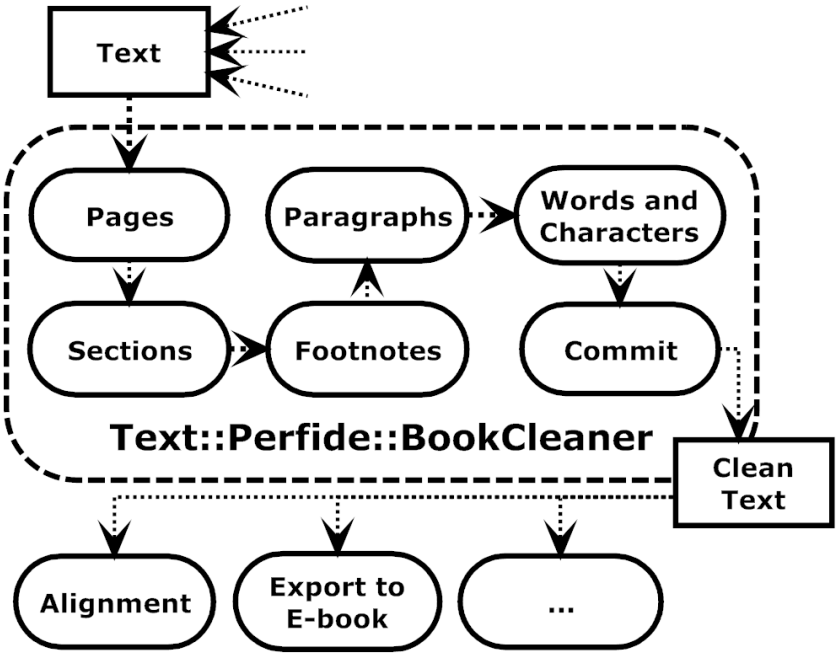


Figure 3.1: Pipeline of `Text::Perfide::BookCleaner`.

3.2.1 Pages

Page breaks, headers and footers are structural elements of pages that must be dealt with care when automatically processing texts, because they represent misleading breaks and elements inserted in the middle of the text. In the particular case of bitext alignment, the presence of these elements is often enough to confuse the aligner and decrease its performance.

The headers and footers usually contain information about the book author, book name and section title. Despite the fact that the information provided by these elements might be relevant for some tasks, they should be identified, marked and possibly removed before further processing.

Example 3.1 presents an extract of a plain text version of *La Maison à Vapeur*, from Jules Verne [Verne, 1880], containing, from line 3 to 6, a page break – in this case, a footer, a page break character and a header, surrounded by blank lines.

```
1 rencontrer le nabab, et assez audacieux pour
2 s'emparer de sa personne.
3
4                    Page 3
5 ^L La maison à vapeur          Jules Verne
6
7   Le faquir, - évidemment le seul entre tous
8 que ne surexcitât pas l'espoir de gagner la
```

Example 3.1: Detail of page structure residues before cleaning.

The `0xC` character (also known as `Control+L`, `^L` or `\f`) indicates in plain text a page break. When present in the text, this character provides a reliable way to delimit pages. Page numbers, headers and footers should also be found near these breaks. There are books, however, which do not have their pages separated with this character.

Our algorithm starts by looking for page break characters and replacing them with a custom page break mark. If none is found, it tries to find occurrences of page numbers. These are typically lines with only three or less digits (four digits would occasionally get year references confused with page numbers), and a preceding and a following blank lines. Then, the algorithm attempts to identify headers and footers: it considers the lines which are near each page break as candidates to headers or footers (depending on whether they are after or before the break, respectively). The number of occurrences of each candidate (normalized so that changes in numbers are ignored) is calculated, and those which occur more than a given threshold are considered headers or footers.

At the end, the headers and footers are moved to a standoff file (only the page break mark is left in the original text).

A cleaned up version of the previous example is presented in Example 3.2.

```
est vrai qu'il fallait être assez chanceux pour
rencontrer le nabab, et assez audacieux pour
s'emparer de sa personne. _pb2_
  Le faquir, - évidemment le seul entre tous
que ne surexcitât pas l'espoir de gagner la
prime, - filait au milieu des groupes, s'arrêtant
```

Example 3.2: Detail of page structure residues, after cleaning.

3.2.2 Sections

There are several reasons that justify the importance of delimiting sections in a document:

- automatically generate tables of contents;
- identify sections from one version of a book that are missing on another version (for example, it is common for some editions of a book to have an exclusive preamble or afterword which cannot be found in the other editions);
- matching and comparing the sections of two books before aligning them. This allows to assess the books *alignability* (compatibility for alignment), predict the results of the alignment and even to simply discard the worst sections (see Chapter 5);
- mining specific sections of scientific manuscripts from the biomedical domain (examples on Section 3.6).

Being an unstructured format, plain text by itself does not have the necessary means to formally represent the hierarchy of divisions of a document, particularly the ones that can be typically found in a book (chapters, sections, etc). As such, the notation used to write the titles of the several divisions is dictated by the personal choices of whoever transcribed the book to electronic format, or by their previous format and the tool used to perform the conversion to plain text. Additionally, the nomenclature used is often language-dependent (e.g. *Part One* and *Première Part* or *Chapter II* and *Capítulo 2*).

Example 3.3 presents the beginning of the first section of a Portuguese version of *Les Miserables*, from Vitor Hugo:

In order to help in the sectioning process, an ontology was built (see section 3.4.1), which includes several section types and their relation, names

```
1 PRIMEIRA PARTE
2 FANTINE
3 ^L LIVRO PRIMEIRO
4 UM JUSTO
5 O abade Myriel
6 Em 1815, era bispo de Digne, o reverendo Carlos
7 Francisco Bemvindo Myriel, o qual contava setenta
```

Example 3.3: Detail of section delimiters before annotation.

of common sections and ordinal and cardinal numbers.

The sectioning algorithm tries to find lines containing section names, pages or lines containing only numbers, or lines with Roman numerals. Then, a custom section mark is added containing a normalized version of the original title (e.g. Roman numerals are converted to Arabic numbers).

The processed version of the example shown previously can be found in Example 3.4.

```
1 _sec+N:part=1_ PRIMEIRA PARTE
2 FANTINE
3 _sec+N:book=1_ LIVRO PRIMEIRO
4 UM JUSTO
5 O abade Myriel
6 Em 1815, era bispo de Digne, o reverendo Carlos
7 Francisco Bemvindo Myriel, o qual contava setenta
```

Example 3.4: Detail of section delimiters after annotation.

3.2.3 Paragraphs

It is often assumed that a new line, specially if indented, represents a new paragraph, and that sentences within the same paragraph are separated only by the punctuation mark and a space. However, when dealing with plain text books, several other representations are possible. For example, some books break sentences with a new line and paragraphs with a blank line (two consecutive new line characters), while others use indentation to represent paragraphs; and there are even books where new line characters are used to wrap text. There are also books where direct speech (from a character of the story) is represented with a trailing dash; others use quotation marks to the same end.

The detection of paragraph and sentence boundaries is of the utmost importance in the alignment process (a wrong delimitation is often responsible for a bad alignment) and in information extraction, as it allows the definition of proper contexts or the extraction of text evidences in support of relationships between named entities.

In order to be able to detect how paragraphs and sentences are represented, our algorithm (1) starts by measuring several parameters, such as the number of words, the number of lines, the number of empty lines and the number of indented lines. Then several metrics are calculated based on these parameters, which are used to understand how paragraphs are represented, and the text is fixed accordingly.

3.2.4 Footnotes

Footnotes are formed by a call mark inserted in the middle of the text (often appended to the end of a word or phrase), and the footnote expansion (i.e. the text of the footnote). The placement of the footnote expansion shifts from document to document, but often it appears either at the end of the same page where its call mark is, or at a dedicated section at the end of the document.

Depending on the notation used to represent footnote call marks, these can introduce noise in further document processing. When aligning books, in the best case scenario, the alignment might work well, but the resulting corpora will be polluted with the call marks, interfering with its further use. Footnote expansions are even more likely to disturb any further processing, as they introduce text which is usually not relevant to the task. They are specially disturbing in the alignment process, as it is very unlikely that the matching footnotes are found at the same place in both books (the page

Algorithm 1: Paragraphs

Input: txt:text of the book
Output: txt:text with improved paragraph

elines ← ...calculate empty lines
lines ← ...calculate number of lines
words ← ...calculate number of words
plines ← ...number of lines with punctuation
$indent_i$ ← ...calculate indentation distrib
plr ← $\frac{plines}{lines}$ // punctuated lines ratio

forall the $i \in dom(indent)$ **do**
  if $i > 10 \vee indent_i < 12$ **then**
    $remove(indent_i)$ // remove false indent
  end
end
wpl ← $\frac{words}{lines}$ // word per line
wpel ← $\frac{words}{1+elines}$ // word per empty line
wpi ← $\frac{words}{indent}$ // word per indent
if $wpel > 150$ **then**
  if $wpi \in [10..100]$ **then**
    Separate parag. by indentation
  end
  if $wpl \in [10..100] \wedge plr > 0.6$ **then**
    Separate parag. by new lines
  end
end
else
  Separate parag. by empty lines
end

divisions would have to be exactly the same).

Example 3.5 presents an example of a page containing footnote call marks and, at the end, footnote expansions, followed by the beginning of another page.

```
roi Charles V, fils de Jean II, auprès de la rue
Saint-Antoine, à la porte des Tournelles[1].

[1] La Bastille, qui fut prise par le peuple de
Paris, le 14 juillet 1789, puis démolie. B.

^LQuel était en chemin l'étonnement de l'Ingénu!
je vous le laisse à penser. Il crut d'abord que
```

Example 3.5: Detail of footnotes before cleaning.

The removal of footnotes is performed in two steps: first the expansions are removed, and then the call marks. The algorithm searches for lines starting with a plausible pattern (e.g. `<<1>>`, `[2]` or `^3`), and followed by blank lines. These are considered footnote expansions, and consequently they are replaced by a custom footnote mark and moved to a standoff file.

Once the footnote expansions have been removed, the remaining marks (following the same patterns) are likely to be call marks, and as such, they are removed and replaced by a custom normalized mark[1]

The results of removing the footnotes from the previous example can be found in Example 3.6.

```
roi Charles V, fils de Jean II, auprès de la rue
Saint-Antoine, à la porte des Tournelles_fnr29_.
_fne8_
^LQuel était en chemin l'étonnement de l'Ingénu!
je vous le laisse à penser. Il crut d'abord que
```

Example 3.6: Detail of footnotes after cleaning.

[1]The ideal would be to be able to establish the correspondence between the footnote call marks and their respective expansion. However, that feature is not specially relevant to the focus of this work, as the effort it would require is not compensated by its practical results.

3.2.5 Words and characters

At word and character level, there are several problems that can affect document processing: Unicode characters, translineations and transpaginations:

Unicode characters: often, Unicode-only versions of previously existing ASCII characters are used. For example, Unicode has several types of dashes (e.g. '-', '–' and '—'), where ASCII only has one (i.e. '-'). While these do not represent the exact same character, they are close enough, and their normalization allows to produce more similar results.

Glyphs: some documents use glyphs (Unicode characters) to represent combinations of some specific characters, like *fi* or *ff*.

Translineation: translineation happens when a given word is split across two lines in order to keep the line length constant. If the two parts of translineated words are not rejoined before processing, they might be seen as two separate words.

Transpagination: transpagination happens when a translineation occurs at the last line of a page, resulting in a word split across pages. However, the concept of page is removed by the first function, `pages`, which removes headers and footers and concatenates the pages. This means that transpagination occurrences get reduced to translineations.

Mathematical formulas: Some documents represent mathematical formulas using Unicode characters, while others use plain ASCII to the same end. Either way, these are difficult to identify and process.

Uncommon scripts: Text written in Cyrillic, Greek characters, etc. are sometimes converted to similar Western characters – α to *a*, $\imath$ to *r*, etc.

Our algorithm for dealing with translineations and transpaginations is available as an optional feature. This is mainly because different languages have different standard ways of dealing with translineations, and many documents use incorrect forms of translineation. As such, the user will have the option to turn this feature on or off. A formal definition of this method can be found in Algorithm 2.

Unicode-characters are dealt with by a simple search and replace. Characters which have a corresponding character in ASCII are directly replaced, while strange characters are replaced with a normalized custom mark (see Algorithm 3).

Algorithm 2: Translineations

Input: txt: text of the book
Output: txt: text with normalized translineations

pat : word chars + dash + line break + word chars `//` translineation pattern

forall the *pat* ∈ *txt* **do**
  remove hifen
  move line break to end of word
end

Algorithm 3: UTF-8 characters

Input: txt: text of the book
Output: txt: text with normalized UTF-8 characters

$UTF8_{chars}$: list of UTF-8 characters with ASCII alternative

forall the *c* ∈ *txt* : *cis a UTF8 character* **do**
  if $c \in UTF8_{chars}$ **then** `replace(`*c,ASCII alternative*`)`
  else `replace(`*c,custom mark*`)`
end

3.2.6 Commit

This last step removes the custom marks introduced by the previous functions and outputs a cleaned document ready to be processed. The only marks left are section marks, which can be helpful to guide further processes.

There are situations where elements such as page numbers are also important (for example, having page numbers in a book is convenient when making citations). However, removing these marks makes the previous steps irreversible. As such, this step is optional, and it is meant to be used only when a clean document is required for further processing.

The details of this step can be found in Algorithm 4.

Algorithm 4: Commit

Input: txt: text of the book
Output: txt: clean text without custom marks

forall the *m* ∈ *txt* : *mis a custom mark* **do**
  if *c* ∈ *section marks* **then** leave it there
  else `remove(`*m*`)`
end

3.3 Diagnostic report

Additionally, after the cleaning process, a diagnostic report is generated, to help the user understand the problems detected in the book, and what strategies were used in order to solve them.

Example 3.7 presents an example of the report produced after cleaning a version of *La maison à vapeur* from Jules Verne:

```
footers = ['( Page _NUM_ ) = 240'];
headers=$VAR1 =
   ["(La maison \x{e0} vapeur Jules Verne) = 241"];
ctrL=1;
pagnum_ctrL=241;
sectionsO=2;
sectionsN=30;
word_per_emptyline=3107.86842105263;
emptylines=37;
word_per_line=11.5658603466849;
lines=10211;
To_be_indented=1;
words=118099;
word_per_indent=52.2793271359008;
lines_w_pont=3263;
```

Example 3.7: Example of diagnostic report.

In Example 3.7 it is stated, among other things, that this document had headers composed by the book title and author name, footers contained the word `Page` followed by a number (presumably the page number) and that the 241 pages were separated with `^L`.

3.4 Declarative objects

The first attempt at writing a program to clean books was a self-contained Perl script – all the patterns and logical steps were hard-coded into the program, with only a few command-line flags allowing the fine tuning of the process. When this solution was discarded for not being powerful enough to allow more complex processing, it was decided to use higher level configuration objects whenever possible. This resulted in the creation of an ontology to describe section types and names.

Declarative objects such as this ontology open the possibility to discuss the subject with people who have no programming experience, allowing them to

directly contribute with their knowledge to the development of the software.

3.4.1 Sections Ontology

The information about sections relevant to the sectioning process was stored on an ontology file [Uschold and Gruninger, 1996]. Currently it contains several section types and their relation (e.g. a *Section* is part of a *Chapter*, an *Act* contains one or more *Scenes*), names of common sections (e.g. *Introduction*, *Index*, *Foreword*) and ordinal and cardinal numbers.

The recognition of these elements is language-dependent. As such, mechanisms to deal with several different languages had to be implemented. Some of these languages even use a different alphabet (e.g. Russian), which lead to interesting challenges.

Several extracts of the sections ontology are presented below. Example 3.8 shows several translations and abbreviations for *chapter*. `NT sec` indicates *section* as a narrower term of *chapter*, and *act* is indicated as a broader term of *scene*.

```
cap
PT capítulo, cap, capitulo
FR chapitre, chap
EN chapter, chap
NT sec
RU глава
```

```
cena
PT cena
FR scène
EN scene
BT act
```

Example 3.8: Extract from the sections ontology: *chapter* (left) and *scene* (right).

`BT _alone` allows to indicate that a given term must appear alone in a line. Numerals have also been included in the ontology, allowing to numerate other terms (Example 3.9).

```
end
PT fim
FR fin
EN the_end
BT _alone
```

```
PT terceiro, terceira,
   três, tres
EN third, three
FR troisième, troisieme
BT _numeral
```

Example 3.9: Extract from the sections ontology: *END* (left) and the *numeral 3* (right).

The typification of sections allows to define different rules and patterns to recognize each type. For example, a line beginning with *Chapter One* may be almost undoubtedly considered a section title, even if more text follows

in the same line (which in many cases is the title of the chapter). On the other hand, a line beginning with a Roman numeral may also represent a chapter beginning, but only if nothing follows in the same line (or else century references, for example, would be frequently mistaken for chapter divisions).

Despite not being fully implemented yet, the relations between sections described in the ontology will allow to establish hierarchies, and, for example, search for *Scenes* after finding an *Act*, or even to classify a text as *play* in the presence of either.

Taking advantage of Perl being a dynamic programming language (i.e. is able to load new code at runtime), the ontology is being used to directly create the Perl code containing the complex data structures which are used in the process of sectioning. The ontology is in the ISO Thesaurus format, and the Perl module `Biblio::Thesaurus` [Simões and Almeida, 2002] is being used to manipulate the ontology.

3.5 Evaluation

The evaluation of a tool such as `Text::Perfide::BookCleaner` might be performed by comparing the results of alignments of texts before and after being cleaned with `Text::Perfide::BookCleaner`.

In order to test `Text::Perfide::BookCleaner`, a set of 20 pairs of books from Shakespeare, both in Portuguese and English, was selected. From the 40 books, half (the Portuguese ones) contained both headers and footers. The books were aligned using `cwb-align`, an aligner tool bundled with IMS Open Corpus Workbench [IMS Corpus Workbench, 1994-2002].

Pages, headers and footers

From a total of 1093 footers present in the documents, 1077 were found and removed (98.5%), and 1183 headers were detected and removed in a similar amount of total occurrences.

Translation units

The alignment of texts resulted in the creation of translation memories, files in the Translation Memory eXchange (TMX) format [Oscar, 2000]. Another way to evaluate `Text::Perfide::BookCleaner` is by comparing the number of each type of correspondence (i.e. translation units) in each file – either 1:1, 1:0 or 0:1, or 2:2 (the numbers represent how many segments were included

in each variant of the translation unit). Table 3.1 summarizes the results obtained in the alignment of the 20 pairs of Shakespeare books.

Table 3.1: Number of translation units obtained for each type of correspondence, with and without using `Text::Perfide::BookCleaner`.

Alignm. Type	**Original**	**Cleaned**	**Δ%**
0:1 or 1:0	8333	6570	-21.2
1:1	18802	23433	+24.6
2:1 or 1:2	15673	11365	-27.5
2:2	5413	4297	-20.6
Total Seg. PT	54864	53204	-3.0
Total Seg. EN	59744	51515	-13.8

The total number of 1:1 alignments has increased; in fact, in the original documents, 39.0% of the translation units were of the 1:1 type. On the other hand, the translation memory created after cleaning the files contained 51.3% of 1:1 translation units. The total number of segments decreased in the cleaned version; some of the books used a specific theatrical notation which artificially increased the number of segments. This notation was normalized during the cleaning process with `theatre_norm`, a small Perl script created to normalize these undesired specific features, which lead to the smaller amount of segments. The number of alignments considered bad by the aligner also decreased, from a total of 10 *"bad alignment"* cases in the original documents to 4 in the cleaned ones.

3.6 Cleaning scientific articles

Despite the previously described methods having been implemented with the specific motivation of cleaning books for alignment, most of them can also be useful in other contexts.

Some of the methods have been adapted to the task of preparing scientific articles for text mining. This resulted in the creation of the prototype of a Perl module, `Text::Perfide::SciPaperCleaner` (`T::P::SPC`), which is largely based on `Text::Perfide::BookCleaner`. The modifications are mostly related to the sectioning process and the sections ontology, the detection of headers and footers and the creation of a function to normalize mathematical formulas within articles.

3.6.1 Sections

Identifying the different sections in scientific articles allows to process them differently in order to achieve different results. For example, information retrieval tends to rely on abstracts whereas information extraction often targets sections such as Materials and Methods and Results. [Agarwal and Yu, 2009];

The basic structure of a scientific paper is often summarized as IMRAD: introduction, methods, results and discussion [Peh and Ng, 2008]. This structure was first used in the 1940s, adopted as a majority in the 1970s and is now the standard for publication [Sollaci and Pereira, 2004]. Often, it transpires to the section names: papers with sections named Introduction, Methods, Results and Discussion are a common occurrence. A quick research has revealed that other common section names a include Background, Context, Motivation, Evaluation, Conclusion(s) and so forth. Nevertheless, not all article sections belong to a well-defined set, and many sections or subsections have a name related to the subject being discussed. In these cases, patterns similar to the ones used in `T::P::BC` were applied.

These conventions in section naming, which are stronger than the ones that exist in the case of books, allowed us to adapt the section ontology created before. The new ontology is also flat: there is no hierarchy between section types because that does not usually happen in scientific articles. The translations of section types into other languages were also removed because most scientific papers are written in English.

3.6.2 Pages

Page structure is similar in books and scientific articles: it is also composed by page breaks, page numbers, headers and footers, which means that the same methods can be applied. However, the methods for identifying headers and footers in books relied on thresholds which cannot be reached in papers, given the fact that the number of pages is considerably lower.

To overcome this question, the identification of headers and footers in articles is performed taking into account the relative frequency of the candidates in the total number of pages. A minimum threshold is still considered, to avoid false positives in particularly short articles.

3.6.3 Normalizing mathematical notation

Whether retrieved in some text-based format or converted from Portable Document Format (PDF) or MS Word formats to plain text, scientific articles

have a wide range of notations used for mathematical formulas. Some documents have their formulas written using just ASCII characters, while others make use of Unicode characters [Sargent III, 2006]. For example, the string `3.9+/-1.5x10(9)-1.7+/-0.5x10(10) cell l(-1)` can be used to represent the range $3.9 \pm 1.5 \times 10^9$ to $1.7 \pm 0.5 \times 10^{10} cell \cdot l^{-1}$ in UTF-8 [2].

A method for normalizing such formulas has been implemented, which applies simple substitution patterns to take care of the most common occurrences, allowing these formulas to be more easily recognized later. Early tests of this prototype have suggested that tasks such as text mining produce better results if the papers are cleaned using this module. Section 8.2 presents some ideas on improvements and extensions foreseen for `T::P::SPC`.

3.7 Summary

This chapter addressed concerns about the types of noise which typically affect automatic text processing, and described the implementation of a Perl module to clean plain text books.

This tool, `Text::Perfide::BookCleaner`, performs the cleaning tasks by tackling one kind of problem at a time, in a sequence of steps which results in a cleaned version of the original document, and generates a diagnostic report which informs the user of what was done.

An additional module, denominated `Text::Perfide::SciPaperCleaner`, was implemented mostly as a demonstration that most of the methods proposed for book cleaning can also be useful to deal with other types of documents, with minor adaptations.

[2] Examples extracted respectively from an XML and a PDF versions of the paper by Limpiyakorn et al. [2006]

Chapter 4

Measuring similarity

The nobleness of life
Is to do thus (embracing), when such a mutual pair
And such a twain can do't, in which I bind,
On pain of punishment, the world to weet
We stand up peerless.

Shakespeare, A Midsummer Night's Dream

This chapter discusses the problem of measuring the similarity between documents, proposing a method based on bag-of-words (BoWs) extracted from text. The application of this method to the tasks of finding near duplicates and candidate pairs is described, along with details of the implementation of a tool to perform such tasks in a pool of files.

4.1 Introduction

When building large text corpora, it is common to find identical or almost identical documents (usually referred to as *duplicates* and *near duplicates*, respectively), specially if the documents are retrieved at large/medium scale from several distinct sources. If not detected, all the copies will be processed, representing a waste of resources [Gong et al., 2008], as well as biasing the corpus, invalidating or at least reducing the accuracy of the subsequent analysis [Kupietz, 2006].

Building parallel corpora, i.e. the alignment of pairs of documents is the perfect scenario for developing and evaluating similarity algorithms. Some methods to find candidate pairs have been proposed [Simões, 2004], but they

are usually focused on documents which come from the same source. The multitude of sources of books in Project Per-Fide meant that a method capable of identifying candidate pairs based solely on the documents contents could achieve better results.

Duplicate and near duplicate finding are tasks which have been addressed before [Seshasai, 2009, Broder, 1997, Kumar and Govindarajulu, 2009]; however, given the existing need to also be able to detect candidate pairs, we have developed a module, `Text::Perfide::BookPairs`, which can be used for both tasks.

4.2 Measuring similarity

Finding (near) duplicates and candidate pairs within a set of documents are both tasks which rely on the definition of *similarity* between two documents.

Books which are translated to another language tend to keep some elements untouched. These elements which are not translated (or whose translation is the same as the original) can be used to calculate the *Jaccard similarity coefficient (JSC)*, a standard measure in information retrieval [Jaccard, 1901, Tan et al., 2006], used to compare the similarity and diversity between two sets. Being A and B two sets, the JSC is given by *the size of the intersection divided by the size of the union* of the two sets, as follows:

$$JSC(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

Translations of a text usually retain some unaltered elements, which do not suffer any change because there is no need to translate them or because they cannot be translated. Examples of these language independent elements (LIEs) are [1]:

Year references: when years are not written in long form, they are the same in many different languages – e.g. "1755" is the same both in Portuguese, English and any other language which also uses Arabic numbers.

Proper names: proper names (character names, names of streets or towns,

[1] In spite of LIEs being, by definition, language independent, languages which use different alphabets have LIEs written differently. Also, some proper names can in fact be translated – for example, the Portuguese city *Oporto* is written as *Porto* in Portuguese, and *Pope Benedict VIII* is written *Papa Bento XVII*. Possible solutions for both problems are presented in Section 8.2.

etc.) are also often left untouched – e.g. "Hamlet" is the same, not matter the language used, as long as it uses the Latin alphabet.

Therefore, the similarity between two files can be calculated by extracting the LIEs and creating a BoW for each file[2], and measuring the JSC of the two BoWs.

4.3 Implementation

A small helper module which implements operations over BoWs was created. This module, `Text::Perfide::WordBags`, includes `file2bag`, a function which, given a file path and a function reference, passes the text of the file to the function, resulting in a BoW – a structure containing the number of occurrences of each word of interest. Additionally, this module contains functions to calculate the cardinality of a BoW and the intersection and reunion of two BoWs.

4.3.1 Extraction of bag-of-words

The first implementation of an algorithm to measure the similarity between books was based on year references. For each book, a set was extracted with all the occurrences of four consecutive digits. Then, the JSC of the two BoWs was calculated, and the similarity values were used to find candidate pairs.

This method seemed to work reasonably well, but it presented some flaws:

- in books with more that 999 pages, page numbers were being confused with year references.
- a reasonable percentage of books had year references in either the header or the footer – usually the year of publication.

The number of occurrences of four consecutive digits in page numbers, headers and footers would be so high that the true positive occurrences would end up being only a small percentage. The fact that some books only had a small amount of year references only aggravated the question.

[2] A bag-of-words is a simplification of a text, where it is viewed as of elements (words) in which order in not important [Harris, 1954]. In this case, a bag-of-word (BoW) consisted of a list with the number of occurrences of each element in the text.

In order to avoid these problems, an extraction of BoWs based on proper names was also implemented, and is now the default option. In this context, however, the concept of proper name has been slightly simplified – are considered proper names all the words which met the following conditions:

- appear in the text beginning with an uppercase letter
- do not appear also beginning with a lowercase letter a significant number of times

A definition of the algorithm developed to extract the BoWs is presented in Algorithm 5.

Algorithm 5: Extracting proper names from text.

Input: txt: text of the book
Output: U_{list}: table of frequencies of proper names

forall the $x \in txt : x$ *is a word* **do**
  lIfx begins with uppercase$U_{list}[x] + +$
  else $L_{list}[x] + +$
end
forall the $u \in dom(U_{list})$ **do**
  $l =$ lowercase version of u
  $u_{count} = U_{list}[u]$
  $l_{count} = L_{list}[l]$
  $ratio = \frac{u_{count}}{l_{count}}$
  if $ratio < 10$ **then** `delete`$(U_{list}[u])$
end

Dealing with Russian books presented additional challenges, as they use a different script and the regular expressions used were not compatible with Cyrillic. This was solved by creating additional patterns which included Russian characters.

Cardinality, union and intersection

Calculation the JSC between two BoWs required that the cardinality of a BoW could be measured, and also that the operations of union and intersection were defined. These operations were defined as follows:

Cardinality: total sum of the occurrences of each word in the BoW.

Intersection: the lower number of occurrences of each word in the two BoWs.

Union: the higher number of occurrences of each word in the two BoWs[3].

4.3.2 Classification according to similarity

Once the concept of similarity was well defined, and the methods to calculate it were implemented, they could be used to find duplicates and candidate pairs. According to their similarity with each other, two documents could be be classified as:

Exact duplicates: Documents which are completely identical to each other, from the first to the last byte.

Near duplicates: Documents which are very similar to each other.

Pairing documents: Documents similar to each other.

Unrelated documents: Documents whose contents are not related.

4.3.3 Identifying exact duplicates

Identifying exact duplicates is a straightforward task, because they present a JSC value of exactly 1. However, very close near duplicates may also have a similarity just as high. To avoid this, the identification of exact duplicates relies on the comparison of the MD5 sum of each file [Rivest, 1992] (although vulnerabilities have been found in the MD5 hashing algorithm [Wang and Yu, 2005], there are no security concerns in this context).

Given that these files are exact copies of each other, only one of the files should be considered for inclusion in the corpus. The copy (or copies, if more that one exist) can be safely deleted without any loss of information.

4.3.4 Identifying near duplicates and candidate pairs

Because near candidates have differences between them, the MD5 sum approach is not applicable, as the sum changes no matter how small the changes might be. Also, both the near duplicates and the pairing candidates usually

[3]The union of two sets is usually a set containing all the elements in the original sets, which means that the union of two identical sets results in a set with every element duplicated. However, from the definition of JSC we have that

$$JSC(A, B) = 1 \text{ when } A = B$$

As such, the result of the intersection and the union between identical BoWs must be the same, and thus the uncommon definition of the union operation.

present high JSC values, which could make it difficult to distinguish between them. However, pairing candidates are, by definition, written in different languages, while near duplicates are written in the same language. This provided us with a possible solution:

- files with a high JSC value and written in the same language are likely to be near duplicates;
- files with a high JSC value but written in distinct languages are likely to be candidate pairs.

The identification of the language in which each file is written is performed by the `Lingua::Identify` Perl module [Simões, A., 2011]. The fact that near duplicates are written in the same language usually also leads to a higher JSC value when compared to candidate pairs. The initial tests allowed to infer that books with a JSC value under 0.2 are probably unrelated; candidate pairs usually present JSC value higher than 0.4 and near duplicates usually have a JSC over 0.9. These have been implemented as the default values but can be overridden.

4.4 Processing a pool of files

Implementing the previously described methods as a tool to process a medium/large sized pool of files presented several challenges of a more practical nature, related with decisions that must be taken when building a corpus.

The methods previously described in this chapter are able to provide an answer to the questions of finding (near) duplicates and finding candidate pairs, but they to not determine what should be done with those documents once they are identified. Exact duplicates can be removed without questions, but what about near duplicates? How to choose which one should be kept and which ones can be discarded?

4.4.1 Command line utility

In order to deal with the problems which are more related with the specific questions of using `Text::Perfide::BookPairs` to classify files, a command-line utility, `pairbooks` was implemented. This program can receive as input:

a file and a list of files, and it compares the first file with every file in the list.

a single list of files, and compares each file in the list with all the others.

two lists of files, and each file in the first list is compared with every file in the second.

By default, it outputs, for each file, a list of the files with the highest similarity value (the size of the list is configurable through a flag). The list can be printed with additional details (the similarity value and the size of each BoWs). An example of such a list can be found in Example 4.1.

```
 ~ $ pairbooks  <(ls -1 PT*) <(ls -1 ES*)
PTBR__Umberto_Eco0_nome_da_rosa.txt
  (0.227) [6954,7382]   ES__Umberto_EcoEl_Nombre_de_la_Rosa.pdf.txt
  (0.018) [6954,11408]  ES__Umberto_EcoEl_Pendulo_De_Foucault.pdf.txt
  (0.018) [6954,5604]   ES__Umberto_EcoDiario_Minimo__2.txt

PTBR__Umberto_Eco0_Pendulo_de_Focault.txt
  (0.391) [11276,11408] ES__Umberto_EcoEl_Pendulo_De_Foucault.pdf.txt
  (0.042) [11276,6024]  ES__Umberto_EcoLa_busqueda_de_la_Lengua_Perfec...
  (0.035) [11276,5604]  ES__Umberto_EcoDiario_Minimo__2.txt

(...)
```

Example 4.1: `pairbooks`: default output.

In order to make it easier to integrate this tool with other alignment-related tools, it is possible to force the output to be in a format which can be directly used by `Text::Perfide::BookSync` and other similar tools. When this option is activated (through a command-line flag), the output contains only the files which are most likely to be real pairs. Several thresholds have been defined, with default values that can be adjusted through command-line options:

Reject threshold (default 0.2): files with similarity below this value will not be considered pairs.

Accept threshold (default 0.4): files with similarity above this value will be considered pairs.

Duplicates threshold (default 0.9): when searching for (near) duplicates, files with similarity above this value will be considered duplicates.

Sometimes there are pairs of files whose similarity value is under the accept threshold. As such, it is possible to optionally output these pairs also, as commented lines, which can be manually checked by the user later. Lines with files whose similarity is under the reject threshold are printed with a

leading `# X` and lines with files with a similarity value above the rejection value but below the accept value will have a leading `# ?`. The output consists of a pair of file names per line, separated by tabs. An example of this output can be found in Example 4.2.

```
 ~ $ pairbooks -bpairs -warn <(ls -1 PT*) <(ls -1 ES*)
PTBR__Umberto_EcoA_ilha_do_dia_ant... ES__Umberto_EcoLa_Isla_Del_Dia_An...
# X  PTBR__Umberto_EcoApocaliptico... ES__Umberto_EcoDiario_Minimo__2.t...
# ?  PTBR__Umberto_EcoO_nome_da_ro... ES__Umberto_EcoEl_Nombre_de_la_Ro...
PTBR__Umberto_EcoO_Pendulo_de_Foca... ES__Umberto_EcoEl_Pendulo_De_Fouc...
# X  PTBR__Umberto_EcoSemiotica_e_... ES__Umberto_EcoLa_estructura_ause...
PTPT__Umberto_EcoBaudolino.txt        ES__EcoBaudolino.doc.txt
```

Example 4.2: `pairbooks`: list of pairs of files, including less probable pairs.

Additionally, the tool can be used to search specifically for near duplicates, in which case it will output files whose similarity is above the duplicate threshold, as seen in Example 4.3.

```
 ~/ $ pairbooks -same -dv=0.7 -v <(ls -1 PT*)
PTBR__Umberto_EcoO_nome_da_rosa.txt.bc_out
(0.842) [6954,7016]  PTPT__Umberto_EcoO_Nome_da_Rosa_Revisto.txt.bc_out
(0.794) [6954,6966]  PTPT__Umberto_EcoO_Nome_da_Rosa.txt.bc_out

PTPT__Umberto_EcoO_Nome_da_Rosa_Revisto.txt.bc_out
(0.842) [7016,6954]  PTBR__Umberto_EcoO_nome_da_rosa.txt.bc_out
(0.759) [7016,6966]  PTPT__Umberto_EcoO_Nome_da_Rosa.txt.bc_out

PTPT__Umberto_EcoO_Nome_da_Rosa.txt.bc_out
(0.794) [6966,6954]  PTBR__Umberto_EcoO_nome_da_rosa.txt.bc_out
(0.759) [6966,7016]  PTPT__Umberto_EcoO_Nome_da_Rosa_Revisto.txt.bc_out
```

Example 4.3: `pairbooks`: list of duplicate files.

4.4.2 Optimization

Extracting BoWs from files and comparing them time consuming, because these tasks involve processing whole books and comparing large data structures. Furthermore, these tasks are usually performed over large collections of documents.

Our first implementation created a BoW for each file each time a comparison between two files was needed. This means that, for example:

1. to discover the near duplicates in a collection with size N, the number of BoWs calculated would be N^2.

2. to find pairs in two sets with size N_1 and N_2, would require $N_1 * N_2$ BoWs to be calculated.

This resulted in too long running times when the size of the sets was moderately high. Some optimizations were then implemented.

The first optimization implemented was a small and simple one: skip any comparison of a file with itself. In fact, while operating over a collection, all files were being matched with all files. Skipping self-comparisons allowed to reduce the number of BoWs created when searching duplicates to $N^2 - N$. This was not too significant, specially with large sets of documents.

The second optimization was to keep the BoWs in memory after they had been created, and reuse them instead of recalculating them. This resulted in an increase on the memory usage, but allowed to reduce the number of BoWs calculated to N in the first case, and $N_1 + N_2$ in the second one.

Finally, we realized that, in the pipeline used Project Per-Fide, `pairbooks` was used in two different steps: first to remove the duplicates, and later to find candidate pairs (possibly more than once if more than one pair of languages was being aligned). The fact that these were different calls to `pairbooks` meant that the BoWs calculated in the first call were erased from memory and not reused to the later calls. To avoid this, another optimization mechanism was implemented: instead of keeping the BoW in memory, it was stored in the disk. The overhead introduced by the writing and reading from disk instead of memory was largely compensated by not having to recalculate all the bags in calls afterwards.

4.4.3 Choosing a version

The question of choosing the best document from all the duplicates does not have an unique answer, as different criteria can be used. In addition to the method described here, other possible methods are discussed in Section 8.2.

Presently, the selection of the version to be kept is being performed based on using a dictionary (in this case, `aspell` [Atkinson, 2006]) to count the number of unknown words in each version. Then the version with the least unknown words is kept, and the others are removed. A more formal definition of this algorithm can be found in Algorithm 6.

Algorithm 6: Selecting a version from a set of duplicates.

Input: dups: files identified as duplicates
Output: file: version which should be kept.

forall the $version_i \in dups$ **do**
  UW_i : unknown words of *version*
  forall the $w \in version : w$ *is a word* **do**
    if $w \notin dic$ **then** `increment`(UW_i)
  end
end
return *version* with the lowest UW

4.5 Evaluation

A set of more than 500 Agatha Christie books in electronic format was put together to evaluate `pairbooks`. The books in this set, Set AC1, were converted to plain text and the duplicates and near duplicates were removed, after which 160 files were left. These files were cleaned with `Text::Perfide::BookCleaner`, and then they were split into groups according to their language, as seen on Table 4.1.

Table 4.1: Number of books in Set AC1, grouped by language.

Language	**Number of books**
Brazilian Portuguese	83
Spanish	33
European Portuguese	35
English	9
Total	160

Then, all the combinations of two languages from Set AC1 were processed by `pairbooks`, i.e. for each pair of languages L1 and L2, `pairbooks` picked up each file from L1 and found its best match in L2. Candidate file pairs with a similarity value above 0.4, were accepted, and the ones with a lower similarity were reported as rejected.

All the candidate pairs (both the accepted ones and rejected ones) were manually validated, and the number of false positives and false negatives was counted. Table 4.2 presents the results of `pairbooks`, the manual corrections

and the precision and recall calculated based on these values.

Table 4.2: Results of `pairbooks` and manual revision.

	Total identified	**Correctly identified**	**Total existing**	**Precision**	**Recall**
Set AC1	99	99	118	1.00	0.84

The manual validation of the results provided by `pairbooks` proved that relying on file names to find the pairs would not have been a good alternative: some files names did not match their contents and, furthermore, the translations of the titles are often very creative, and in many cases the titles of the same book in different languages seem completely unrelated. This required some additional inspection of the files contents and online research to confirm the results.

The results of the evaluation show that `pairbooks` presents a very high precision value, having been correct in every positive pair identification. The recall values are a little lower; however, manually inspecting the rejected pairs has shown that this could be improved by lowering the *accept threshold*: from the rejected candidate pairs, the ones with the highest similarity values were the ones which were in fact true pairs. Lowering the accept threshold to 0.3 would result in a recall value of 0.93, and a threshold of 0.24 would increase the recall up to 0.97. Both would have no influence on the precision value.

This test included books written in two variants of Portuguese, English and Spanish. Undergoing tests which include books written in German and Russian show that these two languages presents some features which cause `pairbooks` to score lower on the precision and recall values. For example, German language capitalizes all the nouns, which causes `pairbooks` to extract too many words from the text.

Some proper names are translated, specially the ones referring to historical characters or locations. Example 4.4 presents the most occurring proper names in several different language editions of *The Name of the Rose*, from Umberto Eco. The number of occurrences and some similarity in the writing allow to identify the translations: for example, Guilherme/Guillermo/William, Abade/Abad, Malaquias/Malaquías/Malachi and Adso/Адсо. The possibility of providing `pairbooks` with a list of correspondences between proper names would allow to minimize this problem.

Russian presents another interesting problem: declensions. These cause the nouns to be written differently depending on number, genre and grammatical function. Example 4.5 presents pairs of similar words appearing on the

	Portuguese	Spanish	Russian	English
1	743 Guilherme	769 Guillermo	175 Малах	796 William
2	299 Abade	299 Abad	147 Берен	181 Malachi
3	232 Deus	236 Dios	145 Хорхе	159 Berengar
4	173 Malaquias	169 Malaquías	119 Венанц	151 Jorge
5	152 Berengário	163 Berengario	114 Бенц	144 Ubertino
6	135 Bernardo	144 Jorge	110 Миха	143 Pope
7	134 Severino	143 Ubertino	92 Адсо	137 Bernard
8	127 Venancio	138 Severino	90 Бернард	131 Severinus
9	121 Jorge	134 Bernardo	84 Убертин	129 Venantius
10	115 Bêncio	131 Adso	82 Бог	117 Benno

Example 4.4: Example of the proper names with higher occurrences in *The Name of the Rose* [Eco, 1980].

top 30 of the list with the proper names with higher occurrences in *The Name of the Rose*. Each pair of words presented is actually the same proper name written in different forms. The problem of declensions could be tackled by implementing a stemming algorithm or by measuring the edit distance [Navarro, 2001] between the highest occurring words.

90 Бернард 29 Бернарда	51 Христа 35 Христос	40 Господь 31 Господа
69 Северин 32 Северина	46 Адельм 38 Адельма	39 Дольчино 38 Дольчина

Example 4.5: Similar words on the top 30 higher occurrences of proper names in *The Name of the Rose*.

4.6 Summary

This chapter described two common problems when performing document alignment: detecting duplicates and near duplicates within a set of files, and finding candidate pairs for alignment.

A method for solving both problems was proposed which allows to identify duplicates and candidate pairs through the measurement of the similarity between files based on LIEs common to both files. This method was implemented in a Perl module, `Text::Perfide::BookPairs` and an auxiliary module, `Text::Perfide::WordBags`.

Then, it was introduced a command-line utility which implements these methods to operate over a pool of files. Several execution options were de-

tailed along with some considerations about the optimizations implemented.

Chapter 5

Synchronizing books

The time rate in the ship and duration on Earth have been unrelated three times. But now they are effectively synchronous again, such that slightly over seventy-four years have passed since we left.

Robert A. Heinlein, Past Through Tomorrow

This chapter introduces the concept of document synchronization, defined as the structural alignment based on section delimitation performed by the module `Text::Perfide::BookCleaner`. A description of its implementation is made, including the generation of visual representations of the synchronization and files with anchor points for alignment.

5.1 Introduction

A common problem when aligning documents is the existence of unmatched sections: entire sections which exist in one version and do not have a match on the other.

Missing sections is a more common occurrence with books than with other types of documents because books are more likely to include sections which are version-dependent (prefaces to a given edition, translator notes, author's biographies and so on). This may however still happen with other kinds of documents – for example, because the document was somehow truncated, or it was only possible to obtain a partial version of it.

The existence of unpaired sections decreases the performance of the aligners, which are usually very sensitive to deletions and insertions, and are not

capable of dealing with such large differences in the texts. The product of such alignments is often too bad to be included in a parallel corpus. Manually correcting the alignment is not a feasible solution when one is dealing with large amounts of documents, and simply removing by hand the badly aligned parts also presents the same problem.

A tool capable of establishing a mapping between the sections of two books and detecting the unpaired sections would allow to identify the problem and act accordingly. Thus, we created `Text::Perfide::BookSync`, a Perl module which implements the functions to detect missing sections in books and align them at section-level – we denominated this structural alignment process as *book synchronization*. This module includes a script, `syncbooks`, which implements the whole workflow.

5.2 Implementation

Book synchronization is a process which takes as input two versions of a given document and builds a mapping between the sections of both versions.

The first step is to compile a list containing the relevant information about the existing sections in each version of the document. This task relies on the annotations introduced by `Text::Perfide::BookCleaner` which mark the location of each section division.

For each section, the following elements are stored:

- section type (if any)
- section number (if any)
- offset of first and last character
- total number of words
- first words within the section

The section alignment is then performed based on this data structure. An example of the structure created can be found on Example 5.1.

5.2.1 Alignment method

The section alignment is performed as follows: each section mark in each book is transformed into a token containing that section's number and type (for example, a mark inserted by `bookcleaner` as `_sec+R: cap=1_` will originate

```
EVIL UNDER THE SUN

Agatha Christie
(...)

_sec+N:cap=1_ Chapter 1

When Captain Roger Angmering built
himself a house in the year 1782 on
the island off (...)

_sec+N:cap=2_ Chapter 2

When Rosamund Darnley came and sat
down by him, Hercule Poirot made no
attempt to (...)
```

```
[ {
    'title' => 'begin',
    'id'    => 'begin',
    'wc'    => '9',
    'end'   => '105',
    'start' => 0
  },{
    'title' =>
          '_sec+N:cap=1_ Chapter 1',
    'id'    => 'cap=1_',
    'wc'    => '4358',
    'end'   => '24378',
    'start' => '106'
  },{
    'title' =>
          '_sec+N:cap=2_ Chapter 2',
    'id'    => 'cap=2_',
    'wc'    => '3895',
    'end'   => '46103',
    'start' => '24379'
  },(...)
]
```

Example 5.1: Excerpt of original text annotated by `Text::Perfide::BookCleaner` (left) and corresponding data structure with section information (right).

the token `cap=1`). The tokens from each book are printed to a file, and the Unix *diff* command is used to compare them.

The *diff* utility [Hunt et al., 1976] uses the Hunt-McIlroy algorithm to solve the *longest common subsequence* problem [Hirschberg, 1975], being capable of comparing two files and discovering the lines that were added or removed between them. By comparing the files which contain the section tokens, we can detect which sections can only be found in one of the original documents. Example 5.2 presents an example of a *diff* file generated from the comparison of two section files.

```
_sec+N:cap=1_ Capítulo I
_sec+N:cap=2_ Capítulo II
_sec+N:cap=3_ Capítulo III
_sec+N:cap=4_ Capítulo IV
_sec+N:cap=5_ Capítulo V
_sec+N:cap=6_ Capítulo VI
_sec+N:cap=7_ Capítulo VII
_sec+N:cap=8_ Capítulo VIII
```

```
_sec+O:cap=1_ Capítulo Primero
_sec+N:cap=3_ Capítulo III
_sec+NA:Fin_
_sec+N:cap=4_ Capítulo IV
_sec+N:cap=5_ Capítulo V
_sec+N:cap=7_ Capítulo VII
_sec+N:cap=8_ Capítulo VIII
```

```
begin     begin
cap=1_    cap=1_
cap=2_  <
cap=3_    cap=3_
        > Fin_
cap=4_    cap=4_
cap=5_    cap=5_
cap=6_  <
cap=7_    cap=7_
cap=8_    cap=8_
```

Example 5.2: Excerpt of the section lists of two books (left and center) and the resulting diff file (right).

Sometimes, two versions of a book have different types for the same sections. For example, one version may be divided in *tomes*, and the other may call it *volumes*; one version may have *chapters* while the other has typeless sections (e.g. sections which are represented only by roman numbers). In

these cases, we have made it possible to perform the section-level alignment based solely on the section numbers, regardless of their type. This way, having different types for matching sections does not prevent two books from being effectively synchronized.

5.2.2 Ghost sections and chunks

By analyzing the output we can assess which sections were only found in one of the versions. However, the fact that a given section was not detected by `bookcleaner` does not necessarily mean that the section is missing, it just means that its beginning could not be found – either because it is actually not there or because `bookcleaner` was not capable of identifying it.

Ghost sections give origin to a problem: what should be done with them?

- They cannot be aligned because, for all practical purposes, they are invisible.
- They cannot be removed along with their matching section either, for the very same reason.
- Removing just the matching section (besides being pointless) would result in an even bigger problem: unpaired ghost sections.

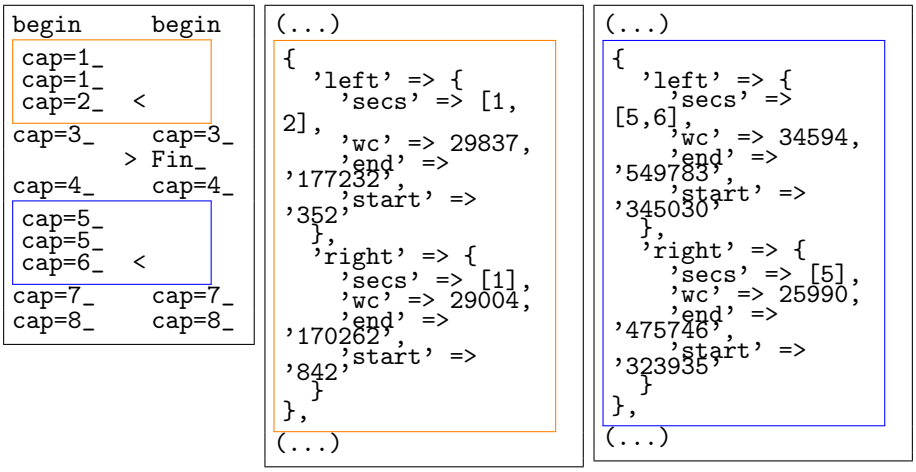
In order to solve this problem, the concept of *chunk* has been created. A chunk consists of a group of consecutive sections, from one version of a book, and a similar matching group from the other version. The relevance of chunks relies on how the number of chunks and which sections belong to which chunk are determined.

Definition 2 *A **chunk** is a data structure which includes a pair of matching sections, and all the following unpaired sections from both documents until the next pair of matching sections, which is the beginning of another chunk.*

$$\textit{chunk: (section} \times \textit{section) (unmatched)* (unmatched)*}$$

◇

Each chunk starts with a pair of matching sections, and includes every following unpaired sections in both versions. Once the next pair of matching sections is reached, a new chunk is created, and the same procedure is followed. An example of the chunks generated from a *diff* file can be found in Example 5.3.



Example 5.3: Diff file (left) and structure with detailed chunk information (center and right). Two chunks have been highlighted in orange and blue.

This means that every matched pair of sections will be at the beginning of a chunk, and every unpaired section will be in a chunk preceded by a matching section. In a perfectly synchronizable pair of books (a pair where every section has one and only one match), each section will be placed on a chunk of its own.

As soon as all the chunks have been determined, the number of words in each chunk is calculated to be further compared.

5.3 Output objects

After all the chunks have been calculated, as well as their size (in number of words), three different output objects can be built: a synchronization matrix, a pair of annotated files, or a pair of sets of split files.

5.3.1 Synchronization matrix

The synchronization matrix consists of an HTML file, built using the Perl module `HTML::Auto`. This matrix contains a visual representation of the synchronization, allowing the user to have a intuitive global perspective on the results of the synchronization.

An example of a synchronization matrix is presented in Figure 5.1. The lines of the matrix correspond to the sections of one of the files, and the

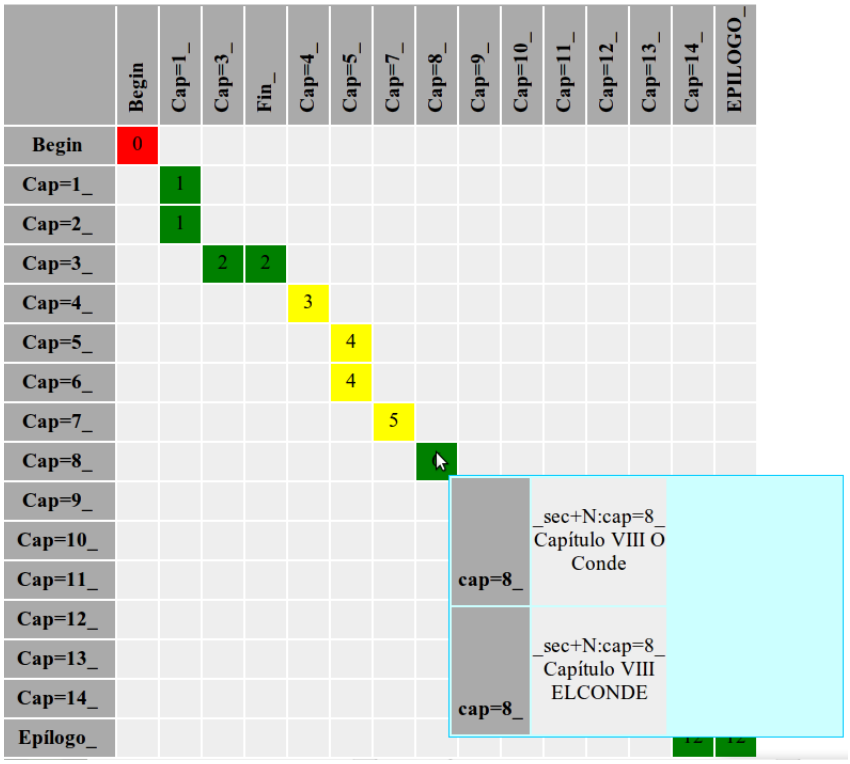


Figure 5.1: Matrix produced as a result of synchronizing the previous examples.

columns to the sections of the other. The numbers in the matrix indicate the chunk those sections belong to. The colors – green, yellow and red – represent how likely it is that the sections in a given chunk really match. This is calculated using the formula

$$L = \frac{wc\_left}{wc\_right} \tag{5.1}$$

where wc_left and wc_right represent the total number of words in the left and right sections of the chunk, respectively. If L is between 0.9 and 1.1, the color green is used; if L is between 0.5 and 0.9 or 1.1 and 1.5, yellow; otherwise, red. Hovering with the mouse over a given square opens a pop-ip containing the first words of each sections. This allows the user to confirm if those two sections have been correctly aligned or not.

5.3.2 Annotated files

Another possible output object consists of a pair of files which are copies of the original input files annotated with *synchronization marks*. For example, for a given pair of files `fileLeft.txt` and `fileRight.txt`, a pair of marked files `fileLeft.txt.sync` and `fileRight.txt.sync` will be created.

The marks are placed at the beginning of each chunk, and they follow the form `<sync id="i">`, where `i` is the number of the chunk. Later on, when the books are being aligned, these marks can be used as *anchor points*.

Frequently, the unpaired sections are found in the beginning of the document: introductions, prefaces, indexes and other introductory segments. As such, an option was added to skip the first `n` chunks, resulting in output files which only start at chunk `n+1`.

Some aligners are not capable of handling large files. In these cases, one possible solution is to split the files in smaller portions. However, the files have to be split in similar ways (i.e. making sure that each pair of smaller files contains the same sections). To this purpose, `syncbooks` is capable of splitting the original files in smaller files, each containing one chunk. This way, the original files `fileLeft.txt` and `fileRight.txt` are split into several `fileLeft.ci` and `fileRight.ci`, where `i` is the number of the chunk contained in the file.

5.4 Summary

This chapter tackled the problem of unmatched sections in documents to be aligned. It introduced the concept of *book synchronization*, a structural alignment of sections, which allows to detect the missing sections before the actual alignment. The implementation of this method, as a Perl module denominated `Text::Perfide::BookSync`, was described, focusing particularly on the section alignment method. The concept of *chunk* was introduced, and its applications were explained.

The several available output objects which can be produced as the result of the synchronization were listed: the synchronization matrix (a visual representation of the synchronization), files annotated with anchor points and pairs of files containing chunks.

Chapter 6

Prototype of a corpora flow

> *It is well. The bad moment has passed. Now all is arranged and classified. One must never permit confusion. The case is not clear yet – no. For it is of the most complicated! It puzzles* me. Me*, Hercule Poirot!*
>
> *– Hercule Poirot*
>
> *Agatha Christie, The Mysterious Affair at Styles*

This chapter introduces the concept of corpora-flow: a workflow for corpora building. A prototype for a corpora-flow system is presented which uses files written in a small defined domain-specific language (DSL) to generate a Makefile which controls the process. This prototype implements several features, namely the workflow control mechanisms created to simplify the task of building a corpus.

6.1 Introduction

Building a corpus is a complex task. One of the many challenges it presents is the fact that it requires each source document to be adapted, processed and transformed by different tools in several consecutive steps whose final result must be ready to be added to the corpus. The tools described so far are meant to help in such tasks, and can be easily assembled into a pipeline where no intermediary steps of conversion are needed.

However, the diversity of documents sources, document types, file formats, notations, encodings, etc. imply that different documents may need different

processing steps in order to reach the same stage. This heterogeneity requires approaches with different levels of sophistication.

The lowest level approach consists of establishing a simple linear pipeline. However, frequently this is of little help, because it is not flexible enough to allow variations of the document workflow. A more sophisticated approach would have to allow to:

- define several alternatives for any given step.
- skip unwanted steps.
- enter and leave the pipeline at any given point.
- define breakpoints where the process holds to allow the inspection of the already performed steps, and if necessary, the intervention of other tools, and resumes its course of action afterwards.
- provide intelligent feedback to the user, while still logging every possible information to allow deeper inspection.

The next subsections will introduce the concept of workflow and describe the general functioning of `make` utilities. Both concepts are used as the basis of our implementation of the corpora-flow prototype.

6.1.1 Workflow

Because it is a concept used in several distinct areas, there are several different definitions of workflow. The Workflow Management Coalition Specification defines workflow as follows [Hollingsworth et al., 1993]:

Definition 3 ***workflow:*** *the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.* ◇

Workflows are commonly used to manage business processes because they allow to abstract the process logic – the higher level rules which command a given process – from the task logic – the steps and operations necessary to perform an individual task.

In the context of this dissertation, a workflow consists of the different steps which define a process: all the alternatives, conditions and sequences of tasks needed to accomplish a given goal. For example, within a project like Project Per-Fide, which aims to build a corpus, the workflow includes

all the tasks which must be performed, from contacting possible sources of documents and getting the permissions to use them to the tasks of preparing the documents, aligning them, evaluating the alignments, etc.

This chapter is focused on the computational tasks of the process of building a corpus, and how to implement a generic system to handle the document processing – in short, the application of the workflow concept to the construction of corpora – **corpora-flow**.

A workflow typically includes the following components:

States: the stopping points of the process.

Actions: the transitions between states.

Pre-conditions: must be verified in order to execute a given action.

Post-conditions: must be verified after executing a given action in order to move to the next state.

Context: the data passed between states and modified by actions.

A workflow can be seen as a graph, in which the nodes are the possible states of the workflow, the edges are the actions, and the pre and post-conditions are annotations at the beginning and end of the edges, respectively.

6.1.2 Makefiles

`make` utilities are tools capable of determining which pieces of a large program need to be recompiled, and issuing the commands needed to do so[1]. Makefiles are the files used by `make` which contain the rules and the dependencies which specify how to derive the target. Example 6.2 presents the basic structure of a Makefile.

Makefiles allow the definition of **macros**, which are commonly used to define variables. A **rule** consists of a dependency line, in which it is stated that one **target** (or multiple targets) depends on a given set of files (**components**). Dependency lines are followed by a series of command lines which define how to transform the components into the target. An example Makefile can be found in Example 6.2.

[1]Several "flavors" of `make` tools are available, which usually present the same basic behavior but can include some specific features. Most of the features discussed in this chapter are common to several of these applications; however, unless stated otherwise, we will be referring to the GNU `make` utility.

```
MACRO = definition
target [target ...]: [component ...]
[<TAB>command 1]
        .
        .
        .
[<TAB>command n]
```

Example 6.1: Basic structure of a Makefile.

```
main.pdf: main.tex
        pdflatex main.tex

clean:
        rm -f main.pdf
```

Example 6.2: Example of a Makefile.

The `make` utility presents several interesting features:

File-oriented: all the rules in a Makefile are triggered by the need to create a target file, and depend on the existence of a set of source files.

Dependencies: dependencies between processes can be easily indicated by establishing dependencies between the intervening files.

Timestamp checking: when the target file already exists, rules will be executed only if one or more of the source files have been edited more recently than the target file.

Fast-fail execution: by default, a `make` command is interrupted as soon as one of the steps exits with a non-zero status, or one of the dependencies could not be satisfied.

Resumable execution: a make process might be interrupted; however, the files which have already been created do not need to be recalculated in a later execution.

Parallelization: `make` is able to parallelize steps which are not linearly dependent on each other.

Patterns: Makefiles allow the definition of rules with target and source patterns which are applied to the files which match the pattern.

The GNU Makefiles use Unix shell commands to described the actions.

`Slay::Makefile` [Nodine, M., 2011] is a Perl module which allows to use Perl code blocks to define the targets, dependencies and the actions in a Makefile.

6.2 Building a workflow with Makefiles

Building a document workflow based on Makefiles allows to take advantage of the Makefile features described in the previous section. Research has revealed the existence of simple uses of Makefiles to control workflows [Dziuba, 2011] and Makeflow, a "workflow engine for executing large complex workflows on clusters, clouds, and grids" based on Makefiles [Albrecht et al., 2011, Thain and Moretti, 2011, Yu et al., 2010].

We propose a workflow system where files written in a small DSL would be parsed and generate a `Slay::Makefile` prepared to manage the corpora-flow. An example of a DSL for corpora-flow is presented in Example 6.3. Simply explained, the DSL states that:

- a workflow is defined by a set of rules.
- each rule includes pre-conditions, an action, and post-conditions.
- an action includes two sets of patterns (which match filenames to be used as targets or dependencies) and a function (Perl code which implements the behavior of the action).
- a condition includes a function (which checks the validity of the condition) and a filename, which is the file that should be created if the condition is true.
- within an action, the pre-conditions prevent the execution of the function.
- within an action, the post-conditions are verified after executing the function, and must be verified in order to create the targets.

In a workflow implemented this way, all the usual workflow components are present: the states are given by the existing targets and dependencies, the actions are implemented as functions, the pre and post-conditions are also implemented as functions and the context is provided by the contents of the files (targets and dependencies).

```
1 workflow:     rule*
2 rule:         pre-condition* action post-condition*
3 action:       targets dependencies function
4 condition:    filename function
5 target:       pattern*
6 dependencies: pattern*
7 function:     Perl code
```

Example 6.3: Example of a DSL for a Makefile-based workflow formal definition.

This implementation also verifies the requirements for a workflow for corpora construction, listed in the previous section:

- the definition of alternative steps can be implemented by defining several rules, each one containing mutually exclusive pre-conditions.
- skipping unwanted steps can also be performed by guarding them with pre-conditions.
- the definition of the entry point in the workflow can be performed by selecting the dependency files available when the `make` command is called.
- the exit point of the workflow is defined by the target selected when the `make` command is called.
- the implementation of interactive breakpoints can be achieved by defining rules whose action calls interactive commands, and rules which depend on files created after the interaction with the user.
- intelligent feedback objects can be built during the execution (for example, by writing a Dot file which generates a graph representing the executed rules of the Makefile).

Chapter 7

Global evaluation

I came here to have you evaluate the proposed procedure. I first wanted to see if the effort would be worthwhile. I see now that it would be.

Isaac Asimov, Utopia

The individual evaluation of some of the tools has already been reported. This chapter, however, presents an evaluation of the whole set of tools, performed with the main goal of assessing the usefulness of the tools in a real use case scenario – preparing books for alignment.

7.1 Alignment evaluation tools

Evaluating the results of alignments is not an easy task. When a gold standard exists, alignments can be evaluated by calculating the precision and recall values. However, when a gold standard is not available, what objective metrics can be used to evaluate alignments? And how to compare two alignments of the same documents, produced under different circumstances (using different aligners, different pre-processing steps and so forth)?

One possible metric for comparison is the number of correspondences of each type. Translations may present phenomena like insertions, deletions and complex correspondences (a sentence split into two or more, or several sentences merged into a single one), which in the alignment originates 0:1, 1:0 or m:n alignments. However, non-1:1 correspondences are frequently a sign that the aligner lost its pace. Two alignments of the same documents can be compared by inspecting the number of correspondences of each type in

each one, and generally the best alignment will be the one which has a higher number of 1:1 correspondences and a lower number of correspondences of other types.

A small set of tools was developed to help in the process of evaluating alignment results. These tools were used both in this global evaluation and also in the evaluation of `Text::Perfide::BookCleaner` (described in Chapter 3), and have been aggregated in the Perl module `Lingua::TMX::Utils` (documented in Appendix A.5).

Inspecting TMXs

`tmx_inspect` is a command-line utility which extracts sample translation units (TUs) from a TMX file, and can be used to quickly check the quality of an alignment – even when an alignment is not perfect (i.e. almost all sentences being correctly aligned), it can still be usable if a great amount of sentences were aligned successfully. Usually, when the results of an alignment are really bad it happened because the aligner lost its pace right near the beginning of the documents.

`tmx_inspect` supports several different criteria for selecting the samples to be extracted from the TMX file:

- A fixed number of samples, equally distributed along the TMX file.
  Example: `tmx_inspect -n=5 file.tmx`
- Fixed points along the file (specified in percentages of the file).
  Example: `tmx_inspect -perc='30,50,80,90' file.tmx`
- Sentences matching a given string or regular expression.
  Example: `tmx_inspect -re='Harry Potter' file.tmx`
- Defining the size of the sample to extract (how many TUs before and after each sampling point).
  Example: `tmx_inspect -ctx=3 -n=5 file.tmx`

An example of a sample of a TMX extracted with `tmx_inspect` can be found in Example 7.1.

Comparing TMXs

`tmx_compare` is a tool which takes as input TMX files and outputs a table with the number of correspondences of each type found in each TMX. This

6 info == 30 % ==	
7 ES ” “ Where have you seen him ?	7 es Yo la he visto . -¿ Dónde la habéis visto ?
8 info == 50 % ==	
9 ES Then , a young man , I did not think of death , but , hotly and sincerely , I wept for my sin .	9 es Pero entonces era joven , y no pensé en la muerte , sino que , copiosa y sinceramente , lloré por mi pecado .
10 info == 80 % ==	
11 ES Do not look around you for a glimpse of the animals of the illuminations you so enjoy on parchments !	11 es ¡ No miréis a vuestro alrededor para ver si descubrís las bestias de las miniaturas con que os deleitáis en los pergaminos !
12 info == 90 % ==	
13 ES You are beginning to reason well .	13 es Empiezas a razonar bien .

Figure 7.1: Example of the output of `tmx_inspect`.

allows to easily observe the changes in the number of correspondences of each type, and thus perform a comparative evaluation of the methods which originated each TMX. An example of the output of `tmx_compare` can be found in Example 7.1.

```
1 ~/ $ tmx_compare tmx1 tmx2
2 PARAM    0:1     1:1    2:1   2:2  Tot_l1  Tot_l2
3 TMX1   15068  101463  14628  7050  147277  142801
4 TMX2   12503  116496  13278  6176  157815  152218
```

Example 7.1: `tmx_compare`: example of output.

7.2 Evaluation process

The evaluation of the tools was performed by using them to pre-process books before alignment. Two sets of books were chosen: Set A comprised Harry Potter books and Set B comprised books written by Umberto Eco. The languages the books were written in included Portuguese, English, Russian, French, Italian, German and Spanish. Two copies of each set were made, resulting in the following configuration:

Set HP1: Harry Potter books, aligned with no previous processing.

Set HP2: Harry Potter books, pre-processed before alignment.

Set UE1: Umberto Eco books, aligned with no previous processing.

Set UE2: Umberto Eco books, pre-processed before alignment.

Books in sets HP1 and UE1 were aligned with no previous cleaning or synchronizing. Books in sets HP2 and UE2 were pre-processed using the set of tools developed in the context of this dissertation (see Figure 7.2):

1. first, the books were cleaned with `Text::Perfide::BookCleaner`.
2. pairs were discovered with `Text::Perfide::BookPairs`[1].
3. pairs were synchronized with `Text::Perfide::BookSync`.

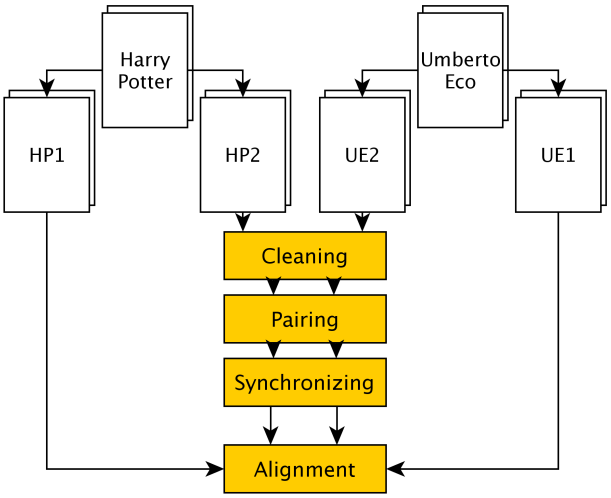


Figure 7.2: Diagram of the global evaluation.

7.3 Results

The tables presented in this section summarize the results obtained in the evaluation process. Table 7.1 details the total number of books aligned in sets HP1 and HP2, the number of those alignments which were considered normal and bad by the aligner[2], the number of alignments which produced no results and the number of alignments which were manually confirmed as being bad. Table 7.2 presents the same details for the the sets UE1 and UE2.

[1]Due to time constraints and to the number of books involver, the pairs in all sets were discovered with `pairbooks`, followed by a manual validation.

[2]The aligned used in Project Per-Fide, `cwb-align`, marks as bad any alignment with a high rate of non-1:1 alignments

Table 7.1: Number of pairs aligned and results (sets HP1 and HP2).

	Set HP1	**Set HP2**	Δ%
Total aligned	25	27	+8.0
Classified as normal	16	20	+25.0
Classified as bad	9	7	-22.2
Missing	2	0	-100.0
Confirmed as bad	9	7	-22.2

Table 7.2: Number of pairs aligned and results (sets UE1 and UE2).

	Set UE1	**Set UE2**	Δ%
Total aligned	67	76	+13.4
Classified as normal	26	48	+84.6
Classified as bad	41	28	-31.7
Missing	9	0	-100.0
Confirmed as bad	39	28	-28.2

In some cases, the alignment process produced bad results, or no results at all. In order to compare the quality of the alignments produced, which was based on the total number of correspondences of each type in each set, it was decided to include only the successful alignments (alignments classified as normal) in the comparison. However, given that sets HP1 and HP2 produced a different number of successful alignments, this would result in comparing sets with different size and contents. To avoid this, it was decided to evaluate *only the alignments which were successful in both sets*. The same is valid for sets UE1 and UE2.

The results of the alignments from books in sets HP1' and HP2' (the subsets from HP1 and HP2 containing only pairs which were successful aligned) were then compared, and the results are summarized in Table 7.3, which presents the number of correspondences of each type found, and the total number of segments (i.e. sentences) in the source language (SL) and the target language (TL). The same evaluation was performed in subsets UE1' and UE2', and the results are presented in Table 7.4.

7.4 Discussion

The results presented in the previous section show that in both test sets the alignment of Harry Potter books has been improved: the number of nor-

Table 7.3: Translation units obtained for each type of correspondence (sets HP1' and HP2').

Corresp. Type	Set HP1'	Set HP2'	Δ%
0:1 or 1:0	12485	7051	-43.5
1:1	104911	124098	+18.3
2:1 or 1:2	22429	19475	-13.2
2:2	3969	3601	-9.3
Total Seg. SL	152569	164302	+7.7
Total Seg. TL	152901	163774	+7.1

Table 7.4: Translation units obtained for each type of correspondence (sets UE1' and UE2').

Corresp. Type	Set UE1'	Set UE2'	Δ%
0:1 or 1:0	14797	10286	-30.5
1:1	202168	207146	+2.5
2:1 or 1:2	26752	23022	-13.9
2:2	7105	6461	-9.1
Total Seg. SL	262200	259268	-1.1
Total Seg. TL	265609	260220	-2.0

mal alignments increased 25%, and the number of bad alignments decreased 22% (Table 7.1). The number of alignments with no results decreased to zero. Alignments which produce no output happen when the aligner quits unexpectedly; there are several causes for this, including running out of memory or catching a file in particular bad conditions.

Even when analyzing the alignments which were successful in both sets, it is possible to observe an improvement: the number of 1:1 correspondences increased 18%, while all the number of other types of correspondences decreased (Table 7.3).

The improvements were even greater in the set of books written by Umberto Eco: The number of successful alignments increased 84.6%, while the number of bad alignments decreased 31.7% (Table 7.2). Additionally, the number of alignments with no results decreased from 9 to 0.

Regarding the results of the successful alignments, there was also an increase in the number of 1:1 correspondences, and a decrease in all other types (Table 7.4).

A deeper inspection of the bad alignments in all sets has revealed that

the majority of them included Russian either as the source language or the target language. German also appeared frequently. This suggests that these languages present some features which make them harder to align. In the case of Russian, the fact that it uses a different alphabet is probably a disturbing factor. The fact that words are frequently concatenated may have been a disturbing factor in the alignment of German books.

Chapter 8

Conclusions and future work

> *"Well, gentlemen," said my friend gravely, "I am asking you now to put everything to the test with me, and you will judge for yourselves whether the observations I have made justify the conclusions to which I have come."*
>
> *– Sherlock Holmes*
>
> *Arthur Conan Doyle, The Valley of Fear*

This chapter presents a summary of the work performed in the context of this dissertation, establishing some conclusions about the usefulness and practical utility of the tools developed and the nature of the problems tackled.

Additionally, future work possibilities are discussed, including specific features and methods which could be added to the existing tools.

8.1 Conclusions

This document introduced several problems which one has to face when building corpora, and described the design and implementation of several algorithms and tools intended to solve them. The tools developed have been implemented as the following Perl modules:

Text::Perfide::BookPairs

Perl module which implements a similarity measuring algorithm, which is applied in functions capable of determining if two documents are duplicates or candidate pairs. Additionally, it includes `pairbooks`, an utility which applies

the modules' functions to process a pool of documents, implementing several optimized methods which make it capable of handling large collections.

Text::Perfide::BookCleaner

Perl module which implements functions to perform cleaning tasks in books – including removing pages structural elements, delimiting sections, removing footnotes, normalizing sentence notation and other types of noise which prevent text documents from being further processed.

Text::Perfide::SciArticleCleaner

Prototype for a Perl module similar to the previous one, but created specifically to clean scientific articles. Functions from `Text::Perfide::BookCleaner` were adapted and additional ones were implemented, such as the normalization of mathematical formulas.

Text::Perfide::BookSync

This module implements algorithms to perform book synchronization – structural alignment at section level – based on the section delimiting performed with Text::Perfide::BookCleane. This synchronization can then be used in order to guide lower-level alignments.

Text::Perfide::CorporaFlow

Prototype for Perl module which allows to implement mechanisms to control the steps included in corpora building workflows – corpora-flows. The workflow control process is based on an extended notion of Makefiles.

The modules developed are publicly available on CPAN[1]. Releasing the tools as open source software required additional work in order to guarantee that the tools could be used and installed in other computers, and that the community standards were followed. It also means that they are continuously being maintained and improved, as more feature requests and bug reports are being issued.

[1]Comprehensive Perl Archive Network – `http://cpan.org`

8.2 Future Work

All the methods and tools described and implemented in the context of this dissertation can be improved and extended. This section lists some features which are planned already or under consideration.

8.2.1 Document cleaners

- The configuration of the existing book cleaner will be performed through a file written in a DSL which will allow to:
  - define the steps to be performed and their sequence.
  - before or after each step, call an external filter to handle specific issues.
  - before or after each step, allow the user to inspect what was done and the current state of the document.
- `Text::Perfide::SciPaperCleaner` will be improved, and cleaners for other types of text can be used (legislation, drama, etc).
- The section ontology will be expanded.
- The algorithm for finding sections will be improved; whenever possible, it will take advantage of the sections hierarchy provided by the ontology.

8.2.2 Document pair finding

- The algorithm for measuring similarity will accept as an additional parameter a list of correspondences, allowing the use of elements other than LIEs to measure the similarity between two documents.
- A machine learning algorithm will be used to check the current default similarity thresholds which are being used to consider two documents as duplicates or as a pair.
- Additionally, a similar algorithm will be implemented to search in runtime for statistical gaps between the similarity values of the documents from a given set, and adjust the default values automatically if necessary.

8.2.3 Document synchronization

- The document synchronization tool `booksync` will be provided with an interactive mode, allowing the user to inspect the results of a synchronization and, if necessary, adjust some parameters before recalculating the synchronization. This will include:
  - *commit* the sections which have been correctly synchronized, and recalculate the remaining ones.
  - select sets of chunks to saved into a file, instead of storing each chunk in a different file.
- The synchronization matrix will be improved to include more information. Other visualization tools will be implemented.
- Better synchronization metrics will be devised and implemented.
- Taking advantage of the section ontology of `Text::Perfide::BookCleaner`, the synchronization algorithm will support hierarchical sections.
- Other synchronization algorithms will be tested, possibly based on the size of the sections (similar to sentence and word-level text alignments).

8.2.4 Corpora-flow

- A first complete version of the DSL for the corpora-flow system will be implemented, as well as the corresponding parser which generates the Slay::Makefiles.
- The Perl module `Text::Perfide::CorporaFlow` will be extended to provide all the necessary functions for implementing a Makefile for the Project Per-Fide workflow.

References

S. Agarwal and H. Yu. Automatically classifying sentences in full-text biomedical articles into Introduction, Methods, Results and Discussion. *Bioinformatics*, 25 (23):3174, 2009. ISSN 1367-4803. **Cited** on page 43.

M. Albrecht, P. Donnelly, P. Bui, and D. Thain. Makeflow: A portable abstraction for cluster, cloud, and grid computing. 2011. **Cited** on page 71.

J.J. Almeida and A. Simões. Automatic Parallel Corpora and Bilingual Terminology extraction from Parallel WebSites. In *Proceedings of LREC-2010*, page 50, 2010. **Cited** on page 20.

S. Araújo, J.J. Almeida, I. Dias, and A. Simões. Apresentação do projecto Per-Fide: Paralelizando o Português com seis outras línguas. *Linguamática*, page 71, 2010. **Cited** on page 5.

S. Atkins, J. Clear, and N. Ostler. Corpus design criteria. *Literary and linguistic computing*, 7(1):1, 1992. **Cited** on page 11.

K. Atkinson. Gnu aspell 0.60. 4, 2006. **Cited** on page 53.

A.Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997. **Cited** on page 46.

P.F. Brown, J.C. Lai, and R.L. Mercer. Aligning sentences in parallel corpora. pages 169–176, 1991. doi: http://dx.doi.org/10.3115/981344.981366. URL `http://dx.doi.org/10.3115/981344.981366`. **Cited** on pages 14 and 16.

L. Burnard. Using sgml for linguistic analysis: The case of the bnc. *Markup Languages*, 1(2):31–51, 1999. **Cited** on page 11.

S.F. Chen. Aligning sentences in bilingual corpora using lexical information. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 9–16. Association for Computational Linguistics, 1993. **Cited** on pages 14 and 16.

Y.C. Chiao, O. Kraif, D. Laurent, T.M.H. Nguyen, N. Semmar, F. Stuck, J. Véronis, and W. Zaghouani. Evaluation of multilingual text alignment systems: the ARCADE II project. In *Proceedings of LREC-2006*. Citeseer, 2006. **Cited** on page 23.

Oxford Dictionaries. "corpus", April 2010. Retrieved in October 27, 2011 from `http://oxforddictionaries.com/definition/corpus`. **Cited** on page 9.

Ted Dziuba. Stupid unix tricks: Workflow control with gnu make. 2011. **Cited** on page 71.

A. Désilets, B. Farley, M. Stojanovic, and G. Patenaude. WeBiText: Building large heterogeneous translation memories from parallel web content. *Proc. of Translating and the Computer*, 30:27–28, 2008. **Cited** on page 23.

U. Eco. *The Name of the Rose*. 1980. **Cited** on page 56.

S. Evert. The CQP query language tutorial. *IMS Stuttgart*, 13, 2001. **Cited** on pages 24 and 30.

W.A. Gale and K.W. Church. A program for aligning sentences in bilingual corpora. *Computational linguistics*, 19(1):75–102, 1993. ISSN 0891-2017. **Cited** on pages 14, 16 and 24.

C. Gong, Y. Huang, X. Cheng, and S. Bai. Detecting near-duplicates in large-scale short text databases. *Advances in Knowledge Discovery and Data Mining*, pages 877–883, 2008. **Cited** on page 45.

R. Gupta and S. Ahmed. Project Proposal Apache Tika. 2007. **Cited** on page 28.

Z.S. Harris. Distributional structure. *Word*, 1954. **Cited** on page 47.

M. Hart and G. Newby. Project Gutenberg. `http://www.gutenberg.org/wiki/Main_Page`, 1997. URL `http://www.gutenberg.org/wiki/Main_Page`. **Cited** on page 14.

D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975. **Cited** on page 61.

D. Hollingsworth et al. Workflow management coalition: The workflow reference model. *Workflow Management Coalition*, 1993. **Cited** on page 68.

A. Hume. A tale of two greps. *Software: Practice and Experience*, 18(11):1063–1072, 1988. **Cited** on page 3.

J.W. Hunt, M.D. McIlroy, and Bell Telephone Laboratories. *An algorithm for differential file comparison*. Bell Laboratories, 1976. **Cited** on pages 3 and 61.

IMS Corpus Workbench, 1994-2002. URL `http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/`. http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/. **Cited** on page 41.

P. Jaccard. Etude comparative de la distribution florale dans une portion des alpes et des jura.[a study comparing the distribution of flora in a portion of the jura alps] bulletin. *Societe Vaudoise des Sciences Natirelles*, 37:547–579, 1901. **Cited** on page 46.

M. Kay. Text-translation alignment. In *ACH/ALLC '91: "Making Connections" Conference Handbook*. Tempe, Arizona, March 1991. **Cited** on pages 14 and 15.

J.D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. Genia corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl 1):i180, 2003. **Cited** on pages 1 and 10.

P. Koehn. Europarl: A parallel corpus for statistical machine translation. 5, 2005. **Cited** on pages 1 and 21.

J.P. Kumar and P. Govindarajulu. Duplicate and near duplicate documents detection: A review. *European Journal of Scientific Research*, 32(4):514–527, 2009. **Cited** on page 46.

M. Kupietz. Near-duplicate detection in the ids corpora of written german. Technical report, Tech. Rep. KT-2006-01. Institut für Deutsche Sprache. ftp://ftp. ids-mannheim. de/kt/ids-kt-2006-01. pdf, 2006. **Cited** on page 45.

P. Langlais, M. Simard, J. Veronis, S. Armstrong, P. Bonhomme, F. Debili, P. Isabelle, E. Souissi, and P. Theron. Arcade: A cooperative research project on parallel text alignment evaluation. 1998. **Cited** on page 17.

K.H. Lee, N. Guttenberg, and V. McCrary. Standardization aspects of eBook content formats. *Computer Standards & Interfaces*, 24(3):227–239, 2002. ISSN 0920-5489. **Cited** on page 30.

T. Limpiyakorn, F. Kurisu, and O. Yagi. Development and application of real-time pcr for quantification of specific ammonia-oxidizing bacteria in activated sludge of sewage treatment systems. *Applied microbiology and biotechnology*, 72(5):1004–1013, 2006. **Cited** on page 44.

Chris A. Mattmann and Jukka L. Zitting. *Tika in Action*. Manning Publications Co., 1st edition, 2011. URL `http://www.manning.com/mattmann/`. **Cited** on page 28.

T. McEnery and A. Wilson. *Corpus linguistics: an introduction*. Edinburgh Univ Pr, 2001. **Cited** on page 11.

I. Dan Melamed. Annotation style guide for the blinker project. *Arxiv preprint cmp-lg/9805004*, cmp-lg/9805004, 1998a. **Cited** on page 19.

I. Dan Melamed. Manual annotation of translational equivalence: The blinker project. *Arxiv preprint cmp-lg/9805005*, cmp-lg/9805005, 1998b. **Cited** on page 19.

R. Moore. Fast and accurate sentence alignment of bilingual corpora. *Machine Translation: From Research to Real Users*, pages 135–144, 2002. **Cited** on page 16.

G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001. **Cited** on page 56.

Nodine, M. Slay::Makefile Perl module, 2011. Retrieved in October, 2011 from `http://search.cpan.org/~nodine/Slay-Makefile-0.12/`. **Cited** on page 71.

D. Noonburg. xpdf: A C++ library for accessing PDF, 2001. **Cited** on page 28.

F.J. Och and H. Ney. Improved statistical alignment models. pages 440–447, 2000. **Cited** on page 24.

T. Okita. Data cleaning for word alignment. In *Proceedings of the ACL-IJCNLP 2009 Student Research Workshop*, pages 72–80. Association for Computational Linguistics, 2009. **Cited** on page 29.

TMX Oscar. Lisa translation memory exchange, 2000. **Cited** on page 41.

WC Peh and KH Ng. Basic structure and types of scientific papers. *Singapore medical journal*, 49(7):522, 2008. **Cited** on page 43.

AM Rassinoux et al. Knowledge representation and management: transforming textual information into useful knowledge. *Yearbook of medical informatics*, page 64, 2010. **Cited** on page 6.

ES Raymond. Basics of the unix philosophy. 2003. **Cited** on page 3.

P. Resnik and N.A. Smith. The Web as a parallel corpus. *Computational Linguistics*, 29(3):349–380, 2003. ISSN 0891-2017. **Cited** on page 15.

R. Rivest. Rfc 1321: The md5 message-digest algorithm. *Status: INFORMATIONAL*, 1992. **Cited** on page 49.

N. Robinson. A Comparison of Utilities for converting from Postscript or Portable Document Format to Text. Technical report, CERN-OPEN-2001, 2001. **Cited** on pages 20 and 28.

D. Santos and P. Rocha. Evaluating cetempúblico, a free resource for portuguese. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 450–457. Association for Computational Linguistics, 2001. **Cited** on pages 1 and 10.

D. Santos and L. Sarmento. O projecto ac/dc: acesso a corpora/disponibilização de corpora. *Actas do XVIII Encontro da Associação Portuguesa de Linguística (APL 2002)(Porto, 2-4 Outubro 2002), APL*, pages 705–717, 2003. **Cited** on pages 1 and 10.

M. Sargent III. Unicode nearly plain-text encoding of mathematics. *Unicode technical note*, 28, 2006. **Cited** on page 44.

S. Seshasai. *Efficient Near Duplicate Document Detection for Specialized Corpora*. PhD thesis, Massachusetts Institute of Technology, 2009. **Cited** on page 46.

M. Simard. The BAF: a corpus of English-French bitext. In *First International Conference on Language Resources and Evaluation*, volume 1, pages 489–494. Citeseer, 1998. URL `http://www.iro.umontreal.ca/~simardm/lrec98/`. **Cited** on pages 21 and 23.

M. Simard and P. Plamondon. Bilingual sentence alignment: Balancing robustness and accuracy. *Machine Translation*, 13(1):59–80, 1998. ISSN 0922-6567. **Cited** on page 16.

A. Simões and J.J. Almeida. Library::*: a toolkit for digital libraries. 2002. **Cited** on page 41.

A. Simões. Parallel corpora word alignment and applications. *Unpublished master's thesis, Escola de Engenharia-Universidade do Minho*, 2004. **Cited** on page 45.

A. Simões and J.J. Almeida. Natools–a statistical word aligner workbench. *Procesamiento del Lenguaje Natural*, 31:217–224, 2003. **Cited** on page 24.

A. Simões and J.J. Almeida. NatServer: a client-server architecture for building parallel corpora applications. *Procesamiento del Lenguaje Natural*, 37:91–97, 2006. **Cited** on page 24.

A. Simões and J.J. Almeida. Parallel corpora based translation resources extraction. *Procesamiento del lenguaje natural*, 39:265–272, 2007. **Cited** on page 24.

Simões, A. Lingua::Identify Perl module, 2011. Retrieved in September, 2011 from `http://search.cpan.org/\begingroup\let\relax\relax\endgroup[Pleaseinsert\PrerenderUnicode{вЇё}intopreamble]ambs/Lingua-Identify-0.30/`. **Cited** on page 50.

J. Sinclair. Corpus and text-basic principles. *Developing linguistic corpora: A guide to good practice*, pages 1–16, 2005. **Cited** on pages 9 and 11.

L.B. Sollaci and M.G. Pereira. The introduction, methods, results, and discussion (IMRAD) structure: a fifty-year survey. *Journal of the Medical Library Association*, 92(3):364, 2004. **Cited** on page 43.

M.Q. Stearns, C. Price, K.A. Spackman, and A.Y. Wang. Snomed clinical terms: overview of the development process and project status. In *Proceedings of the AMIA Symposium*, page 662. American Medical Informatics Association, 2001. **Cited** on page 1.

P.N. Tan, M. Steinbach, V. Kumar, et al. *Introduction to data mining*. Pearson Addison Wesley Boston, 2006. **Cited** on page 46.

D. Thain and C. Moretti. Abstractions for cloud computing with condor. *Cloud Computing and Software Services*, page 153, 2011. **Cited** on page 71.

J. Tiedemann. Word alignment-step by step. pages 216–227, 1999. **Cited** on page 23.

J. Tiedemann. Word to word alignment strategies. page 212, 2004. **Cited** on page 23.

J. Tiedemann. Building a multilingual parallel subtitle corpus. *Proc. CLIN*, 2007. **Cited** on page 14.

J. Tiedemann. Lingua-Align: An Experimental Toolbox for Automatic Tree-to-Tree Alignment. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'2010)*, 2010. **Cited** on page 21.

J. Tiedemann, L. Nygaard, and T. Hf. The OPUS corpus–parallel and free. In *In Proceeding of the 4th International Conference on Language Resourcesand Evaluation (LREC)*. Citeseer, 2004. **Cited** on page 14.

Y. Tsvetkov and S. Wintner. Automatic Acquisition of Parallel Corpora from Websites with Dynamic Content. In *Proceedings of LREC-2010*, 2010. **Cited** on page 15.

UnmarkedSoftware. TextSoap: For people working with other people's text, 2011. **Cited** on page 29.

M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, 11(02):93–136, 1996. **Cited** on page 40.

D. Varga, P. Halácsy, A. Kornai, V. Nagy, L. Németh, and V. Trón. Parallel corpora for medium density languages. *Recent Advances in Natural Language Processing IV: Selected Papers from RANLP 2005*, 2005. **Cited** on pages 14, 24 and 30.

J. Verne. *La maison à vapeur*. 1880. **Cited** on page 31.

V. Vincze, G. Szarvas, R. Farkas, G. Móra, and J. Csirik. The bioscope corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC bioinformatics*, 9(Suppl 11):S9, 2008. **Cited** on page 1.

J. Véronis. From the Rosetta stone to the information society. pages 1–24. 2000. **Cited** on pages 1, 15 and 28.

J. Véronis and P. Langlais. Evaluation of parallel text alignment systems. volume 13, pages 369–388. 2000. **Cited** on pages 17, 21 and 23.

X. Wang and H. Yu. How to break md5 and other hash functions. *Advances in Cryptology–EUROCRYPT 2005*, pages 19–35, 2005. **Cited** on page 49.

M. Wynne, Languages AHDS Literature, and Linguistics (Organization). *Developing linguistic corpora: a guide to good practice*. Oxbow Books on behalf of the Arts and Humanities Data Service, 2005. **Cited** on page 11.

R.Z. Xiao. Well-known and influential corpora. 2008. **Cited** on page 10.

L. Yu, C. Moretti, A. Thrasher, S. Emrich, K. Judd, and D. Thain. Harnessing parallelism in multicore clusters with the all-pairs, wavefront, and makeflow abstractions. *Cluster Computing*, 13(3):243–256, 2010. **Cited** on page 71.

Appendix A

Software Documentation

This section includes documentation for the commands and modules developed Notice that these tools are being developed continuously: so, for up-to-date documentation check the current versions available at CPAN – `http://search.cpan.org`.

A.1 Software Installation

The Perl modules presented in this document are available at CPAN – `http://cpan.org`, and, as such, can be installed with any of the CPAN modules installation utilities – `cpan`, `cpanm`, `cpanp`, etc.

Alternatively, they can be manually downloaded from CPAN, compiled and installed. More information in included in the INSTALL file contained in the package of each tool.

A.1.1 Requirements

At the moment, these tools only work on Unix systems, and depend on:

- A modern version of Perl (above 5.8)
- Unix tools like grep, diff, sort, etc
- Several Perl modules, installable through CPAN.

A.2 bookcleaner

Prepare books for alignment and other operations

Synopsis

```
bookcleaner [options] file*

bookcleaner [options] file.dbooks
```

Description

Prepare a textual book (or a list of books in a file with the extension "dbooks", with one book path per line) for future align operations. The following steps are done:

Step1 – pages, headers footers

Step1 – pages, headers footers (-p1=0 to skip this step)

Step2 – sections

Step2 – sections (-p2=0 to skip this step)

Step3 – paragraphs

Step3 – paragraphs (-p3=0 to skip this step)

Step4 – footnotes

Step4 – footnotes (Deactivated by default. -p4=1 to perform this step.)

Step5 – char level cleaning

Step5 – char level cleaning (-p5=0 to skip this step)

Commit

Commit

Options

```
-c  Commit at the end (removes several debug marks (_pb, etc) before creating output fi

-j=1c    Just do step 1 and commit
-j=...p  Just ... and send output to STDOUT

-simplify     to do several char level simplifications:
```

```
    translate some CP1252 chars to unicode
    translate several dashes, quotes and double quotes  to ascii
    defaul=1
    use -simplify=0 to avoid simplification

-v=34  Create temporary output files of the step 3 (file.ou3) and 4 (file-ou4)

-minhf=3 removes headers or footers if they appear more than 3
     times (def:5)

-pipe   send output to STDOUT

-latin1

-o=FILE send output to FILE (default is original file with extension bc_out)

-dry   Dry run (DEBUG option, makes bookcleaner  do  nothing and just output
               the names of the files received as input

-dir=DIR create all output files under DIR/
```

AUTHOR

Andre Santos

J.Joao Almeida, jj@di.uminho.pt

See also

perl(1).

Text::Perfide::BookCleaner(3pm)

Ontology sections.the

A.3 pairbooks

For a given book, finds its most probable pairs in a collection

Synopsis

```
pairbooks [options] book candidates*

pairbooks [options] book_list1 book_list2
```

Description

Options

```
1  -nr=3          Returns the 3 most similar candidates

2  -bpairs    Output results in .bpairs format (1 pair of books per
3                         line, separated with a \t

4  -rv=LOW    Reject value - book pairs with pairability value
5                         lower that LOW will be automatically rejected.
6                         Default is 0.2

7  -av=HIGH   Accept value - book pairs with pairability value
8                         equal or above HIGH will be automatically approved.
9                         Default is 0.4

10 -dv=VAL        Duplicates value - book pairs with pairability value
11                        equal or above VAL will be considered duplicates
12                        (the same book in the same language).
13                        Default is 0.9. Use with -same.

14 -warn          Comments pairs with pairability value under HIGH
15                        (see -av).
16                        Rejected pairs will have a leading '# X', while
17                        dubious pairs will have a leading '# ?'.

18 -same          Instead of finding pairs, tries to find candidates
19                        to be *the same book in the same language* (see -dv)

20 -debug         Prints debug information

21 -recalc        Calculate file.bag even if it exists already.

22 -normbf        Do not remove .bag files at the end

23 -v
```

AUTHOR

Andre Santos, andrefs@cpan.org

J.Joao Almeida, jj@di.uminho.pt

See also

perl(1).

A.4 syncbooks

Synchronizes books based on section marks produced with Text::Perfide::BookCleaner

Synopsis

```
1  syncbooks [options] file.bpairs

2  syncbooks [options] file1 file2
```

Description

Synchronizes two books (file1 and file2) or several pairs of books, passed in a file with extension "bpairs", each pair in one line with names separated by tab.

Options

```
1  -split  splits file1 and file2 in numbered files (chunks) where each
2      file1.lXXX matches file2.rXXX

3  -mark inserts synchronization marks <sync id="..."> and generates
4      file1.sync and file2.sync. This is the default.

5  -rm=n do not output the first n chunks to the sync files
6      (use with -mark)

7  -noclean  do not remove any sections marks left after synchronizing (default is to remo

8  -html   treate a C<teste.html> file with alignment matrix.

9  -num   ignore section type, use only section numbering to align

10 -dump generate file with Dumper from secs and chunks (debug only)
```

AUTHOR

Andre Santos, andrefs@cpan.org

See also

perl(1).
Text::Perfide::BookCleaner(3)

A.5 Lingua::TMX::Utils

The great new Lingua::TMX::Utils!

VERSION

Version 0.01

Synopsis

Provides functions for analyzing TMX files.

```
1    use Lingua::TMX::Utils;

2        my $tmx_file = 'file.tmx';
3        tmx_inspect($tmx_file);

4        my $tmx1 = 'file1.tmx';
5        my $tmx2 = 'file2.tmx';
6        print_tmx_cmp(tmx_cmp($tmx1,$tmx2));
```

EXPORT

```
1    tmx_cmp

2    print_tmx_cmp

3    tmx_inspect
```

SUBROUTINES/METHODS

tmx_cmp

tmx_cmp($tmx1,$tmx2,...);

For each TMX file passed as argument, measures the number of segments in each type of alignment (0:1/1:0, 1:1, 1:2/2:1, 2:2) and the total number of segments in each variant.

print_tmx_cmp

print_tmx_cmp(tmx_cmp($tmx1,$tmx2,...));

Pretty prints the results of tmx_cmp()

print_tmx_cmp

print_tmx_cmp($tmx); print_tmx_cmp($tmx,$n);

For a given $tmx file, prints $n*2 (default $n=5) sample lines of the $tmx at 30, 50, 80 and 90% of the file.

AUTHOR

Andre Santos, <andrefs at cpan.org>

BUGS

Please report any bugs or feature requests to bug-lingua-tmx-utils at rt.cpan.org, or through the web interface at http://rt.cpan.org/NoAuth/ReportBug.html?Queue=Lingua-TMX-Utils. I will be notified, and then you'll automatically be notified of progress on your bug as I make changes.

SUPPORT

You can find documentation for this module with the perldoc command.

```
perldoc Lingua::TMX::Utils
```

You can also look for information at:

- RT: CPAN's request tracker (report bugs here)
  http://rt.cpan.org/NoAuth/Bugs.html?Dist=Lingua-TMX-Utils
- AnnoCPAN: Annotated CPAN documentation
  http://annocpan.org/dist/Lingua-TMX-Utils
- CPAN Ratings
  http://cpanratings.perl.org/d/Lingua-TMX-Utils
- Search CPAN
  http://search.cpan.org/dist/Lingua-TMX-Utils/

ACKNOWLEDGEMENTS

LICENSE AND Copyright