# MLT-prealigner: a Tool for Multilingual Text Alignment

**Pedro Carvalho and José João Almeida**

**Departamento de Informática, Universidade do Minho**
**Braga, Portugal**
{pedrocarvalho,jj}@di.uminho.pt

## Abstract

Parallel text alignment is a key procedure in the automated translation area. A large number of aligners have been presented along the years, but these require that the target resources have been pre-prepared for alignment (either manually or automatically). It is rather normal to encounter mixed language documents, that is, documents where the same information is written in many languages (Ex: manuals of electronic devices, touristic information, PhD thesis with dual language abstracts, etc).

In this article we present MLT-prealigner: a tool aimed at helping those that need to process mixed texts in order to feed alignment tools and other related language systems.

**1998 ACM Subject Classification** I.7.2 Document Preparation

**Keywords and phrases** parallel corpora, multilingual text alignment, language detection, Perl, automated translation

**Digital Object Identifier** 10.4230/OASIcs.SLATE.2014.283

## 1 Introduction

With the rising importance of multilingualism in language industries, parallel corpora, which consists of source texts along with their translations into other languages, have become a key resource on natural language processing and, in special, the translation area.

Parallel text alignment is a necessary step so that we can generate all sorts of data based on these texts. Although we can easily find a number of tools that already tackle this subject (like Hunalign [10] and Giza++ [5]), it is still rather difficult to automatically align texts in two or more different languages when they are all contained in the same document.

While working on the Per-Fide project [1] we stumbled upon various sources where files like the ones referred above were encountered and were considered useless unless some kind of preprocessing was made on them. In many of these files the texts and their translations where divided in similar fashion. That lead us to catalog a number of patterns that could be easily found and dealt with. After this identification and cataloging process, we reached the conclusion that a generic solution could be proposed to each and every one of these patterns. That was the main motivation that lead to the creation of this tool.

MLT-prealigner is intended to be used as a preprocessor that can consume documents like the ones referred above and render them usable by any classical aligner. It is intended to separate these documents into multiple files and also to tag the output, according to their language, paving the way for any further processing.

This tool was made to tackle the documents that follow the patterns that we will thereafter present, and although it is not a *one size fits all* solution we think that it is a much needed help to multilingual text alignment and to the translation area in particular.

3rd Symposium on Languages, Applications and Technologies (SLATE'14).
Editors: Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões; pp. 283–290

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Outline.** In the following pages we state the problem more clearly in Section 2. The tool's architecture will be presented in Section 3. Case Studies and results can be viewed in Section 4. Finally a brief discussion will be made in Section 5.

## 2 Background

The problem of language identification has been address (with good results) in a large number of articles and tools. However some specific problem constraints and details raise the necessity of building new solutions or new variants. In this world we can find problems like: (i) classification of small units (e.g. one line sentences); (ii) the use of minority, or poorly covered, languages (e.g. Tetum and its variants); (iii) documents with (large amounts of) language independent words (e.g. Entities, Person names, bibliography). This paper will focus on a specific problem, multilingual documents.

When gathering resources for multilingual text alignment for our different projects, documents with multiple translations contained in them were a common sight. Usually, these documents followed certain patterns and it was easy to see that dealing with them was a repetitive task. As such, these patterns were cataloged, and served as a starting point for MLT-prealigner's core idea. The most common patterns were:

- Documents where the languages are all in sequential sections that do not intertwine (example: abstracts in thesis, articles, etc.), but there is no clear separation point between them. This pattern is called *contiguous sections*.

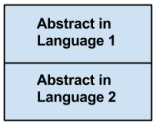


**Figure 1** Contiguous sections in different languages pattern.

- Documents where the languages are separated by clear markers. The two most common markers were pages and paragraphs. So, we cataloged two cases, *zipped pages* and *zipped paragraphs*.

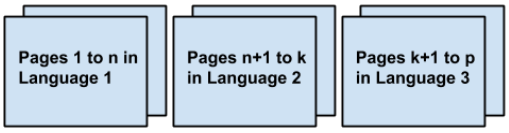


**Figure 2** Zipped paragraphs/pages pattern.

- Documents where the languages are contained in columns, different languages in different columns (we call this *zipped columns*).

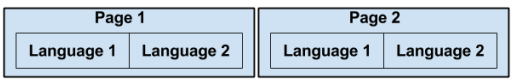


**Figure 3** Zipped columns pattern.

Some of these examples are discussed in Section 4.

In the present, we have access to publicly available good quality language identifying tools, such as: (i) command line tools that have a very good response for most cases, provided they have a minimum document length (e.g. 15 words); (ii) web services (SOAP, REST) such as Open Xerox [11]; (iii) libraries for all the popular programming languages, (e.g. `Lingual::Identify::CLD` [9] for Perl); (iv) works like King et al. [4] presented promising results on mixed-language documents.

Although we use generic tools (such as `Lingua::Identify::CLD`) to do some language classification activities, we realized that it was useful to build a project-specific module in order to have better control on the configuration details. That lead to the creation of our tool, presented in the following section.

## 3 MLT-prealigner Architecture

We took a dictionary based approach using the `Text::Aspell` Perl module, and public available dictionaries from the Aspell project and similar. Our intention was to create a tool that was able not only able to separate languages that are within multilingual documents in an automated manner (with a high level of precision), but also easily customizable, so that the user has more control.

Problems regarding document format will be disregarded in this paper. We have built some inside-solutions, but the focus of this paper will be upon the extracted text and its manipulation. Experience showed that in some situations, it is useful to use this tool in collaboration with cleaning tools (ex bookcleaner [7, 8]) to remove some document noise (ex: page numbers, headers and footers, etc).

### 3.1 Work-flow

The process is sequential. Our input file passes first through our *splitter* component, which divides the file's content into *smaller chunks* and then redirect the file to the *tagger* component. That component will analyze each chunk and try to identify which language it is written in, it will then tag the chunk accordingly. The next step is creating the output files based on the *tagged chunks*, that job is done by the *unzipping* component, which will create a new file for every language that has been identified, pasting in it the chunks that belong to that language.

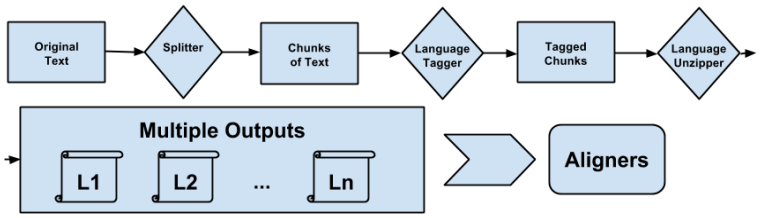


**Figure 4** MLT-prealigner's work-flow.

### 3.2 Splitting

We observed that the splitting logic could be cataloged in two major families. Firstly we have the case where just applying out splitting action would lead to an estimate of the languages of our resulting chunks. Good examples for this kind of splitting are documents with zipped

**Require:** text chunk and dictionaries list
1: break the chunk in words
2: **for all** words **do**
3:  **for all** dictionaries **do**
4:  **if** word is contained in dictionary **then**
5:  increment dictionary's counter
6:  **end if**
7:  **end for**
8: **end for**
9: **if** a dictionary has a definitive advantage over the others **then**
10:  **return** dictionary's language
11: **else**
12:  call tiebreaker algorithm
13: **end if**

**Figure 5** Main algorithm for language detection.

pages or zipped columns, if we know beforehand that page 1 is written in language $L_1$ and page 2 is in language $L_2$, then splitting both pages should be sufficient to extract both translations. This means that in the next step of our work-flow only a language verification is needed instead of trying to identify it from scratch. In these cases, usually, the separation point between languages is very clear and should be used as our *splitting target*. Our second splitting family is, naturally, the antitheses of our first case. In cases where there is no possibility to estimate the languages of the resulting chunks (there is no clear separation point between languages) then a more generic split has to be applied, and a full language identification has to be made. At the moment, the splitting strategy is manually defined.

## 3.3 Language Detection and Tagging

MLT-prealigner uses a dictionary based approach to language detection. The main principle is cross checking words against dictionaries; these dictionaries may be user defined and it is possible to build a dictionary from a list of words, this case is very useful for minority languages, like Tetum, where it is very difficult to obtain a robust spell-checker dictionary. Our implementation of this component uses `Text::Aspell` [2] Perl module.

### 3.3.1 Main Algorithm for Language Detection

As said above, the main principle of the algorithm is cross checking words against dictionaries. When a dictionary has a high enough number of words contained in it, it's language will be selected as the language of the chunk of text that is being processed. In the case that no specific dictionary was able to breakthrough then another algorithm will be applied, to these algorithms we gave the name *Tiebreaker algorithms*. At the time this article was written it was still a bit unclear at which point we could safely say that a specific dictionary had *broken through*, or even if this threshold should be user defined. A rough sketch of the algorithm is presented in Figure 5.

### 3.3.2 Tiebreaker Algorithms

Because there is the possibility that the main algorithm may not be able to detect, with a high degree of confidence, the language of all chunks of text, we had to envision certain ways that we could do a deeper inspection of the text so that a decision could be reached. Several tiebreaker algorithms were proposed and as we found out, the success percentage of these algorithms directly depended on the language division pattern of the document that was being processed. So one algorithm could be more suitable for one type of problem, but when in another context another algorithm would be more suited for that specific problem. By

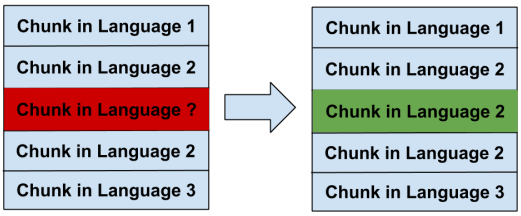


**Figure 6** Language scheme after the algorithm.

giving the user direct control on what tiebreaker algorithm to apply, we are also giving the tool a new level of flexibility.

**Neighborhood Algorithm.** In certain patterns, the fact that a chunk of text is surrounded by other chunks, that share the same language between them, is a clear indicator that this chunk in itself may be written in that language as well. So if we can detect that the immediate predecessor and successor of the chunk in question are both written in the same language, then we assume that the chunk itself belongs to that language (Figure 6).

This type of reasoning is very successful in cases where we know before hand that different languages won't mix, instead there are specific sections for each language in the document that is being processed. One such case will be addressed in Section 4.1.

**Reversed Neighborhood Algorithm.** Just as the name indicates, this algorithm is the exact opposite of the former one. In cases where we previously know that the chunks of text will be intercalated according to their language, we can detect that the immediate predecessor and successor are of *Language 1* and so the chunk of text that is being analyzed is of *Language 2*. This reasoning is suitable for cases like *zipped paragraph*. A case study that tackles this pattern will be presented in Section 4.3.

**Dimension Algorithm.** In some cases it is expected that the resulting chunks in each identified language do not vary a lot in size. To verify if this similarity in size between the chunks is as expected a metric was defined. Considering $\sigma$ as our *size function*.

$$max(\sigma(L_1, L_2)) \geq min(\sigma(L_1, L_2)) \times \delta$$

where $\delta$ is the *size coefficient* that is defined as our threshold, for example, if $\delta = 2$ then the biggest chunk was expected to be, at most, twice as big as the smallest one.

This kind of reasoning can very well be applied in cases as *zipped pages* and *zipped columns*, as long as the number of the target pages or columns is more or less equal. When the output does not match the expected dimensions, then a finner analysis has to be made, this means reducing the set of active languages and/or decreasing the size of the text chunks.

## 3.4 Output Generation

After the original text has been split and all the chunks tagged, the next step is to produce the output files. MLT-prealigner allows for two types of output generation, the process that we call *unzipping* and also the creation of a *tagged file*.

We call *language unzipping* to the creation a new output file for each language that has been detected. The text chunks are distributed accordingly to the tag that has been previously inserted.

The other type of output generation that is available is the creation of a file with all the tagged chunks. This is specially useful when our aim is not text alignment in itself, instead we may want to do another kind of processing, like obtaining statistics about the chunks. A small example of one of this type of file can be seen below.

> `[en]` The classic crown ethers are macrocyclic polyethers that contain between 3 and 20 oxygen atoms, separated from each other by two or more carbon atoms. `[/en]`
>
> `[pt]` Os éteres coroa clássicos são poliéteres macrocíclicos que contêm 3 a 20 átomos de oxigénio, separados entre si por dois ou mais átomos de carbono. `[/pt]`

We ended up using this notation instead of XML due to potential problems that could arise due to the appearance of nested XML tags inside our documents. To identify each chunk we used the same nomenclature as the Aspell dictionaries. For instance, standard Portuguese will be tagged with `[pt_PT]` and Brazilian Portuguese with `[pt_BR]`[1].

## 4 Case-studies and Results

Because this tool was created pretty much out of necessity, as we encountered several text alignment issues when trying to undertake certain projects, it seemed interesting and relevant to present some of the problems we faced and were solved using MLT-prealigner. In the next subsections we will discuss these problems and how we proceeded to solve them.

### 4.1 Abstracts of Rcaap Thesis

The Portuguese Open Access Scientific Repository (Rcaap)[6] is home to a large number of academic thesis. This type of document usually has an abstract translated in one or more languages. Knowing this, we undertook a small project with the intention of creating a *parallel text corpus* based on the collection of abstracts available on the repository, which were very rich in terms of technical terminology, however approximately 31% of these abstracts were contained in single documents with all the translations mashed up together, one after another, just as in Figure 1.

To be able to use these documents we had to find a way to split the chunks of text according to its languages, originating a new document for each language detected, in the case of the example above, 3 documents would be produced. The main barrier was to find the separation point between languages so we could divide the text into multiple documents.

We chose to split the texts by *sentence*, and because we knew before hand that the languages would come one after another, it seemed that this was a pattern suitable to use the *neighborhood algorithm* as the tiebreaker.

We made a small analysis on the outputs, 50 random files were chosen, and realized that 1043 sentences were processed and 13 of them were mistakenly tagged with the wrong language, this gives us a 98.75% success percentage. A big part of those errors were due to citations inside the text, very common in academic texts like this, where, for example, the expression “et al”, which is very common in citations, would almost always make MLT-prealigner think the sentence was in French.

[1] The full list of dictionaries can be found in `ftp://ftp.gnu.org/gnu/aspell/dict/0index.html`

## 4.2 Multilingual Manuals of Electrical Appliances

In this case-study we describe the separation of a set of PDF multilingual manuals of electrical appliances of Teka[2]. In near 80% of the available manuals, we found several different languages in the same PDF document. In some situations we had just partial translations.

We noticed the existence of multilingual title-pages after processing the entire corpus. Some problems were created by imperfect PDF to text conversion of texts that included images and other non textual elements.

Observing just the Portuguese Spanish pair, we obtained 273 in each language of length varying from 2 to 39 pages. After the alignment process, we obtained 43 alignments marked as "bad-alignment" and 230 that passed the automatic quality check assessment of the project.

## 4.3 PT-Tetum Bilingual Version of "O anjo de Timor"

In this case-study we discuss the separation of a bilingual text: "O anjo de Timor" [3]. The book is a bilingual document (PT, Tetum). Tetum is a minority language, and therefore not covered by `Lingua::Identify::CLD` [9], neither Aspell dictionaries. The multilingual chunks were separated into paragraphs. Also, pages included headers and footers.

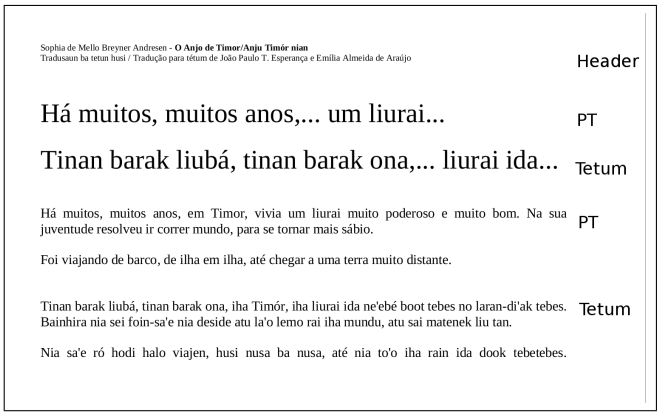


**Figure 7** Example bilingual document.

## 4.4 Results

This tool is in a very preliminary phase, and so evidence of it's utility is still scarce. However, building on the aforementioned case studies, some results were calculated, and our first results are very encouraging. At the moment this paper was written, unfortunately, results on the *Anjo de Timor* case study were not calculated.

# 5 Conclusions

We believe that this tool will have a big impact in the gathering of new resources, a task that is critical in multilingual text processing, especially in the translation area.

[2] `http://www.teka.com`

**Table 1** Preliminary results.

| Case-Study | Dimension | Precision |
|---|---|---|
| Rcaap | 1043 sentences | 98.75% |
| Teka (PT-ES) | 2800 pages | (very good lang detection 98%; some PDF problems) |
| Anjo de Timor | to be calculated | |

As new patterns are identified, and new solutions are engineered to deal with them, the tool will grow and become even more useful, that is why one of the main qualities of MLT-prealigner is that it's tiebreaker algorithms set is easily extensible.

We envision that the one of the next steps in the development of the tool will be to build some kind of multilingual text classifier to automatically detect patterns.

Another key point of improvement, is alternating it's dictionary based approach for language detection with an approach based in `Lingua::Identify::CLD`. Because both approaches have their *pros* and *cons*, we believe we may be able to take the best of both worlds.

Finally, the possibility to create robust dictionaries for minority languages, based for example on word lists, is something that should be a great asset to the translation area.

## References


1 José João Almeida, Sílvia Araújo, Nuno Carvalho, Idalete Dias, Ana Oliveira, André Santos, and Alberto Simões. The Per-Fide corpus: A new resource for corpus-based terminology, contrastive linguistics and translation studies. In Tony Berber Sardinha and Telma São-Bento Ferreira, editors, *Working with Portuguese Corpora*, chapter 9, pages 177–200. Bloomsbury Publishing, April 2014.

2 Kevin Atkinson. Aspell spell checker, 2011. `http://aspell.net/`.

3 Sophia de Mello Breyner Andresen. *Anjo de Timor, Anju Timór nian.* Instituto Camões, 2003. Bilingual edition Tetum-Portuguese, translated by J. Esperança and E. Araújo.

4 B. King and S. Abney. Labeling the languages of words in mixed-language documents using weakly supervised methods. In *NAACL-HLT*, 2013.

5 F. J. Och and H. Ney. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 440–447. Association for Computational Linguistics, 2000. Giza++.

6 Rcaap. Project Rcaap – repositório científico de acesso aberto de portugal. (home page), FCT, 2010. `http://www.rcaap.pt/`.

7 André Santos and José João Almeida. Text::Perfide::BookCleaner, a perl module to clean and normalize plain text books. In *Actas del XXVII Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural*, 2011.

8 André Santos, José João Almeida, and Nuno Carvalho. Structural alignment of plain text books. In Nicoletta Calzolari et al., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

9 Alberto Simões and Google Chrome Team. `Lingua::Identify::CLD` – Perl interface to Google Chrome Language Detection library, 2013. `http://search.cpan.org/dist/Lingua-Identify-CLD/`.

10 D. Varga, P. Halácsy, A. Kornai, V. Nagy, L. Németh, and V. Trón. Parallel corpora for medium density languages. *Selected Papers from RANLP 2005*, pages 590–596, 2005.

11 Xerox. Open xerox language identifier system, 2013. `http://open.xerox.com/Services/LanguageIdentifier`.