Grabbing parallel corpora from the Web

José João Almeida jj@di.uminho.pt
Alberto Manuel Simões albie@alfarrabio.di.uminho.pt
José Alves de Castro jac@natura.di.uminho.pt
Departamento de Informática
Universidade do Minho

Resumen: Los recursos multilingües son útiles para los estudios lingüísticos, para la traducción y para muchas otras tareas. Sin embargo, estos recursos son difíciles de obtener y de organizar. En este documento describimos un conjunto de herramientas diseñadas para ayudar en la tarea de extraer recursos bilingües de la Red que sirvan para construir corpora paralelos y memorias de traducción. Nuestro objetivo es construir herramientas que puedan ser compartidas o usadas de manera independiente.
Palabras clave: Corpora paralelos, extracción de la Red

Abstract: Multilingual resources are useful for linguistic studies, translation, and many other tasks. Unfortunately, these resources are difficult to obtain and organize. In this document we describe a set of tools designed to help in the task of mining bilingual resources from the web, from a specific site, from a file system, from a list of URLs, or from a translation memory. As a design goal we intend to build tools that can be used both cooperatively (in pipeline) and also in a independent way.
Keywords: Parallel Corpora, web-mining

1 Introduction

Text parallelization and parallel corpora is important for natural language processing, language studies, dictionary creation and can be easily transformed on translation memories.

For this, we need to have parallel texts, but it is difficult to find them. In this document we present a web based approach to retrieve potential parallel texts from the world wide web, classify, and create parallel corpora with them.

Figure 1 shows a general view of the process we will present. Each down arrow number are respective to the following items:

$$
\begin{gathered}
web \vee directory \\
\Downarrow_1 \\
(File^2)^\star \\
\Downarrow_2 \\
(File^2)^\star \\
\Downarrow_3 \\
((p^\star \times id)^2)^\star \\
\Downarrow_4 \\
((p \times p)^\star \times id^2)^\star
\end{gathered}
$$

Figure 1: General View of the Process

1. To retrieve texts from the Internet we need to detect text translations in diverse languages, and to make it correspond interdependently. In section 2 we present two ways to get these texts: using Resnik Parallel Strands, or using a new approach, named Parallel Guessing.

2. After the potential parallel texts have been fetched, a process for validation is started. It will check the probability for these texts to be parallel and keep only valid ones. This process uses auxiliary tools which will be discussed in sec. 3.

3. In the corpora building stage we split parallel texts into sentences and convert them to a PML[1] file.

4. Follows the alignment itself. In this task, we can generate translation memories (TMX[2]) back and forth.

To make these corpora useful, we decided to use the IMS CWB to maintain them. In section 4.3 we show how this tool works, and how we can build CGI scripts to consult the parallel corpus.

While this process can be used to produce the corpora itself, our intent is not only the material preparation but also to construct a

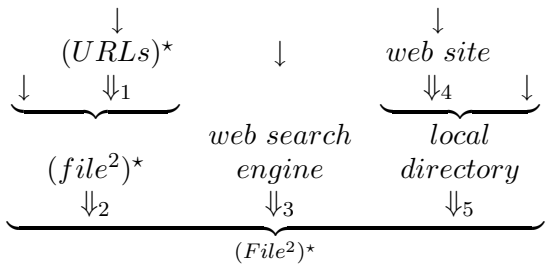
[1] Paragraph Markup Language — a XML document with only one element, the paragraph one
[2] Translation Memory eXchange

set of tools to perform various tasks independently: align text files which came from different sources than the web, or extract parallel information from the web without making a real corpus with it.

On section 5 we present our evaluation for the algorithms and techniques we use.

2 *Retrieval of candidate pairs*

The origin of a candidate pair doesn't need to be local. This diagram illustrates the various processes to retrieve candidate pairs.



Simple arrows symbolize entry points (the points we can start from). Double arrows are processes described in the next five subsections:

1. get file pairs from a list of URLs;
2. using user supplied file pairs;
3. get file pairs from the result of queries on search engines;
4. trying to guess parallelism from a web site, consisting of getting them as local files, and then using process 5;
5. trying to guess parallelism from a set of directories.

These different approaches are further analyzed in the next subsections.

2.1 Extracting candidate pairs from a list of URLs

Given a list of URLs (created by a web robot, for example), that list can be processed to form blocks of similar URLs, in other words, create the congruence class for a normalization function. This function removes language tags (for English, for example, these tags are `english`, `eng` and `en`[3]) and converts all letters to lower case. After doing this for each one of them, we will have blocks of similar URLs associated to their common part. This is a very fast and reliable process.

[3]as in ISO 639(ISO 639, 1992).

An example of this process would be the one of associating the two following URLs, associating to their common part[4]

```
http://www.ex.pt/index_pt.html
http://www.ex.pt/index_en.html

http://www.ex.pt/index_.html
```

After this step, each block containing at least two entries is processed to retrieve possible pairs from it (i.e., considering the two languages being searched, each two URLs where each of them contains a tag for one of the languages).

The resulting list of candidate pairs is then processed as described in section 2.2, as if it were a list provided by the user.

2.2 Processing a list of filename pairs

As the process shown in the previous section creates a list of candidate pairs, this can also be created by another program or by the user. The list contains in each line the pathname of two files to be compared, separated by a [TAB].

2.3 Mining the web for candidate pairs

The algorithm used for web retrieval of candidate pairs is based upon the one of Philip Resnik(Resnik, 1998).

First, a query is submitted to a web search engine (currently Altavista); the purpose of this query is to find pages with at least two hyperlinks, each one of them mentioning one of the names of each of the languages being searched. Let's say we're looking for candidate pairs in Portuguese and English. Web search engines like Altavista accept queries in the form[5]:

```
    (anchor:"portuguese" OR anchor:"portugues")
AND (anchor:"english" OR anchor "anglais")
AND NOT "dictionary"
```

In the next step of candidate generation, each page returned by the web search engine is automatically processed to extract all valid pairs of URLs (i.e., pairs of hyperlinks, each

[4]note that the normalization process takes place only on the directory structure of the URL, not in the domain name

[5]The `AND NOT "dictionary"` part is intended to discard dictionaries, which are not what we are looking for.

one of them mentioning a name of a language currently being searched, both of them pointing to files of the same known extension).

The candidate pairs are then retrieved from the web validated and compared as described in section 3.

2.4 Extracting candidate pairs from a site

Given the URL for a site, that site is retrieved from the web via the GNU Wget (Niksic, 2001), and stored in a directory. That directory is then browsed as described in section 2.5.

From the set of options Wget provides, we have chosen the ones that would enable recursive retrieving following relative links only, not ascending to the parent directory, nor downloading a file more than once.

2.5 Extracting candidate pairs from a directory

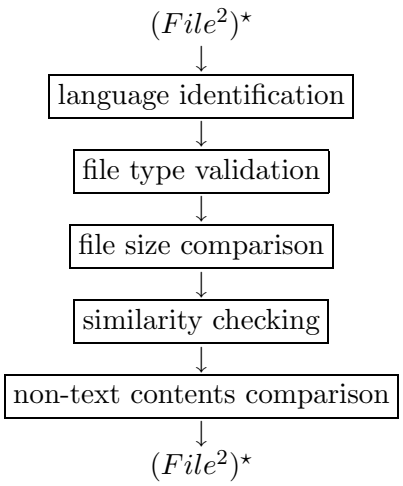
Given a directory, the first step consists of searching the list of files in its tree and applying the same process done for URL lists (described in section 2.1). If the number of candidate pairs found is not large enough, we follow to the next step. Given the directory, a list of all of its subdirectories to a chosen level is created. Then, for each pair of directories (including reflexive pairs), if at least one of the following conditions is verified[6], all possible combinations of files in the first directory are tested with files in the second directory:

- to be the same directory, within a certain depth (for the value 2, ¡a/b,a/b¿ but not ¡a/b/c,a/b/c>);
- to be in the same depth, and to have a common parent in a certain distance (for the value 1, ¡m/a,m/b¿ but not ¡m/a/b,m/c/d¿);
- one of them being a parent of the other, within a certain depth (for the value 2, ¡d,d/a/b¿ but not ¡d,d/a/b/c¿);
- being on different levels within a certain distance, and having a common parent in a certain distance from the deepest (for the values 1 and 2, ¡d/a,d/b/c¿ but not ¡d/a,d/b/c/e¿ or ¡d/a/f,d/b/c/e¿).

[6] actually, the four conditions presented reflect the four main structured ways to store multilingual information in the web (and possibly on other sources); with the correct choice of options for these conditions, one can easily reduce the number of candidate pairs without loss of true translation pairs.

3 Validation of candidate pairs

This part of the process works like a filter. It receives a list of file pairs and will check if they really are as parallel texts. Next diagram illustrates these operations:



The validation of candidate pairs is divided in three major parts. First, the files are tested for language identification; next, some file type validation is done; finally, an in-depth validation is performed. These three steps are described in the next subsections.

3.1 Language identification

Since none of the ways to provide candidate pairs is certain of giving a pair of files in the languages being searched, a module for language identification was built.

This module contains implementations of the "small word technique" and the "trigram method"(Grefenstette, 1995), as of prefix and suffix techniques, among others.

It also includes a main method that chooses the algorithm to use according to the input text (regarding its size, its number of words and some other factors), resulting in a quite reliable and quick process.

If the candidate pairs are not in the desired languages they are immediately discarded. Some of the previous methods to find candidate pairs can improve performance using language identification by reducing the amount of tests.

3.2 File type validation

The module for file comparison extracts (from the files extensions) the types

of the files and compares them with its known types, validating pairs like ¡f1.htm,f2.html¿ but not ¡f1.txt,f2.pdf¿ or ¡f1.xpto,f2.xpto¿ (which is of an unknown format).

3.3 In-depth validation

This in-depth analysis consists of checking the file sizes, names, final punctuation (`!`, `?` and `.`) and non-text contents.

File size comparison is explained in section 3.3.1. The filenames are normalized as explained in section 3.3.2 and compared for similarity with the algorithm described in section 3.3.3. The algorithm for similarity between punctuation strings is the same. The non-text content comparison is described in section 3.3.4.

3.3.1 File size comparison

The comparison of file size consists on checking the percentage of space of the bigger file that the smaller one occupies. The mathematical formula for this is:

$$sizecomp(f_1, f_2) = \frac{smaller\ file\ size}{bigger\ file\ size}$$

For this comparison to be made a cleaned copy of the file is created; i.e., for each file, everything that is not considered content (natural language) is discarded. So, for HTML files, for example, everything inside a tag is not important.

If we had not discarded tags for HTML files, pages with no natural language information would be considered; this problem is not restricted to this file type.

3.3.2 String normalization

String normalization is used to compare filenames, URLs and non-text contents. It consists on removing extensions, converting all letters to lower case, removing language prefixes, infixes and suffixes (as "pt" for Portuguese or "en" for English) and removing any non-letter character.

For a string as `Index_pt.html`, this process would transform it to `index`. This has the purpose of equalizing strings like `Index_pt.html` and `index-en.htm`, for example.

3.3.3 Similarity checking

In order to calculate the similarity between two strings, we determine the edit distance and divide it by the maximum possible distance there could be between them.

The edit distance between two strings is the amount of simple operations (insertion and removal of characters) needed so that the first will match the second.

The resulting number has to be multiplied by 100 and inverted (subtracted from 100), in order to give the expected percentage[7].

$$similarity(a, b) = 1 - \frac{edit\ distance(a, b)}{max\ possible\ dist(a, b)}$$

The edit distance is calculated with an existing perl module, named `String::Approx` (Hietaniemi, 2001). The function is used twice (one for the distance of the first to the second and other in the opposite direction) because it is not commutative.

As an example, when comparing the strings `wwwuminho` and `wwwdiuminho` (these strings have already been normalized), the edit distance from the first to the second is 2 (two characters must be inserted), and the distance from the second to the first is also 2 (two characters must be removed). The total distance is 4.

The maximum distance from a string to another is the sum of their lengths (removing all characters from the first and inserting the ones from the second).

For this example, the result would be 90%.

If we were comparing `wwwuminho` and `wwwoxford` (which are not much similar at all), the result would have been of 33 percent.

The formula used can be written as:

$$1 - \frac{adist(str_1, str_2) + adist(str_2, str_1)}{2\ (length(str_1) + length(str_2))}$$

where `adist` is the function of `String::Approx` that calculates the edit distance between two strings.

3.3.4 Non-text content

The next step is to compare non-text content in the files, such as links and images; they are extracted using regular expressions which are part of the program configuration. A simplified regular expression for extracting HTML links would be:

[7] following formulae calculate values between 0 and 1. This is to make them smaller to fit the two column format.

```
<a\s+href\s*=\s*"(.*?)"\s*>
```

The program accepts a configuration which includes expressions like this one; so, for HTML files, for example, we have provided expressions for links and images.

For each of these expressions, the program tries to match it inside each file. Then, the part inside brackets is normalized (as described in 3.3.2) and the following result is calculated:

$$\frac{similar\ matches}{total\ number\ of\ matches}$$

This result is a percentage and is used in the calculation of the final value, described in section 3.3.6.

3.3.5 Minimum results required

If the results of size comparison are too small (for example, 10 percent), we can immediately discard the candidate pair without continuing its validation[8].

With this in mind, we have provided the option of configuring a minimum value for each of the heuristics being used (these values are dependent of file type).

This process has two good sides. First, it helps eliminating non-translation pairs. Second, the process for comparing two files will immediately stop when confronted with a too low value, thus reducing the processing time involved.

3.3.6 Final results

With the results of size, filename, punctuation and non-text content comparisons, the final value for the comparison between two files can be achieved. This is done with a weighted mean, configurable and independent for each file type.

$$validation(f_1, f_2) = \\ \textbf{let} \begin{cases} a &= sizecomp(f_1, f_2) \\ b &= similarity(f_1, f_2) \\ c &= similarity(punct(f_1), punct(f_2)) \\ d &= content(f_1, f_2) \end{cases} \\ \textbf{in}\,\{\, weighted\ mean(a, b, c, d)$$

If the resulting value is bigger than a chosen percentage (which is independent for each file type), the files are considered to be translations of each other.

[8] some studies indicate that, for true translation pairs, this value would be of at least 80 percent (this value may vary according to specific languages).

4 From file pairs to parallel corpora

The parallelization units should not be the full text, because we are talking about big text portions. So, we split the text in small chunks. This process is file type dependent. For example, HTML will be transformed into a Paragraph Markup Language (PML) file only with paragraph (`p`) tags. At the moment, this process is done to all types of structured documents. First, we convert the original type to HTML and then to PML. For plain text files, we convert them directly to PML.

The PML files will be aligned by each resulting paragraph. This alignment can produce translation memories (in `tmx` format) or be sent to a corpora management program like CQP.

4.1 From HTML to PML

The conversion from HTML to PML is done using structured document processing. We define three sets of tags: tags to remove, but process structurally the contents (`id` set), tags to remove including the contents (`remove` set), and tags to transform to paragraphs (`transform` set).

This way, we can define sets like:
$id = \{body, html, font, a, b, i, tt, small\}$
$remove = \{head, meta, img\}$
$transform = \{td, p, br, li, dt, dd, h1 \ldots\}$

Notice that we are trying to break by sentences or, at least, coherent units. If we would like to align by word, it could be a better idea to transform tags like `a` or `b` to blocks.

For each tag t found in the source document we will check:

- $t \in id$: return the processed contents;
- $t \in remove$: return nothing;
- $t \in transform$: return a `p` tag with the processed contents.

The resulting paragraphs will be processed by a sentence detector. This tool will split each paragraph into sentences and put each one of them in a different '`p`' tag. In section 4.3 we will present some more information regarding this tool.

4.2 From text to PML

This tool uses empty lines to detect paragraphs. On other cases, it is configurable to

break lines as paragraphs. Notice that we are speaking of text files and in some cases, we have complete paragraphs with more than one sentence on a big line. In these cases, we must break the file by new lines.

As in the previous section, each paragraph will be processed by the sentence detector to split them on smaller chunks of text.

4.3 `Lingua::PT::pln`

For many of the natural language processing tool tasks, we have a common base of tools we use everywhere. This lead us to make a Perl module to include all common functions.

One common tool is a sentence detector. Taking text as input, this tool uses some heuristics to detect if punctuation marks are sentence delimiters or only used for abbreviations, `URL`'s or e-mails.

4.4 From XML to CWB

The `xml2cwb` was built to make CWB corpus from a XML file. In order to do that, we need to:

1. extract the list of used tags and make the list of the used attributes for each tag, and build the registry file (in the CWB specific format);
2. make the tokenization of the files and convert it to the CWB format (one token per line with the attributes separated by tabs);
3. execute the appropriate IMS-CWB commands – `cwb-encode` and `cwb-makeall` with the appropriate options.

`xml2cwb` can accept the following options:

-lema to calculate the list of possible lemmas and part of speech (POS) for each token in the corpus, in order to make the query expressions more powerful. To add the lemma and POS information, we are using jspell morphological analyzer(Almeida and Pinto, 1994; Simões and Almeida, 2001);

-quebra this option turns on extra sentence separation.

4.5 From TMX to CWB and back

As we said one of our main goals is make the information reusable. In order to do that we want to have a rich set of translators to import/export from different formats.

Translation Memory Exchange (TMX) is SGML based standard used in machine translation world to store translation memories (TM) – a set of translation unit (for example, sentences) pairs.

Generally, machine translation tools like Trados(Trados, 1998-2002) or Déjà Vu(Déjà vu, 1993-2002) accept and export TM in TMX format. With the `tmx2cwb` and `cwb2tmx` commands, we want to be able to import/export TM to our system. With these commands we can export parallel corpora to TMX, edit with TMX tools and import it again.

4.6 PML pairs alignment

The alignment process is done using two special files. We do not align each file pair by itself, as the alignment process is not as quick as we would like. Instead, for each set of file in some language, we create another XML file in which we will put `file` tags and inside them, the `p` tags for that file:

EasyAlign(IMS Corpus Workbench, 1994-2002) is a text alignment tool that is a part of IMS CWB tool kit. It tries to align chunk of text by size, non-text content and other heuristics. Because we know each alignment side is a sequence of files, we can force synchronization in each `file` tag in a way file contents will not be aligned to different files.

The following pseudo-code shows the mutual alignment process.

```
1  PMLalig(f1,f2)=
2    c1 = xml2cwb(f1)
3    c2 = xml2cwb(f2)
4    easyAlign(c1,c2,
5              unit=>"p",
6              syncWith=>file:id)
7    easyAlign(c2,c1,
8              unit=>"p",
9              syncWith=>file:id)
10   [add align information to c1 registry]
11   [add align information to c2 registry]
```

4.7 Generic CGI for CWB corpora

To consult corpora, we have developed a CGI which provides methods to search a normal or a parallel corpora. If it is a parallel one, it shows two columns, each one with one language (see figure 2).

The interface makes it possible to search for words, and get only the first n matches for that one. Then, you can ask for context information, like k words each side of the searched

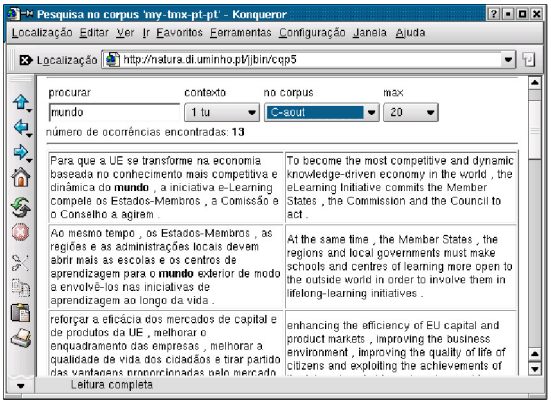


Figure 2: Querying the extracted corpus with a CGI

one, or to show the full sentence, or the full translation memory, for example.

5 *Evaluation and results*

For the purpose of evaluation, we have experimented with two different approaches.

The process of these experiments and the respective results are described in the next subsections.

5.1 First experiment — extracting candidate pairs from a site

For the first experiment, we have downloaded an entire site and then randomly selected a portion from it, consisting of 15 files in Portuguese and 19 files in English. The approach used was the one described in section 2.5, thus comparing all files among themselves.

We have configured the program so that only HTML files would be considered, and fixed the values for size comparison, name similarity, punctuation similarity, links and images, all as 10. As for the required similarity for a pair of files to be considered a parallel translation, we have set that value to 75 percent.

We have also decided that the required minimum values for the various heuristics would be 50 for for size comparison and 0 for all the others.

total number of files	34
files in Portuguese	15
files in English	19
true translation pairs	12
files identified as Portuguese	12 (80%)
files identified as English	16 (84%)
number of comparisons done	192
number of possible comparisons	561
pairs found	13
correct pairs found	11
precision	11/13 = 85%
recall	11/12= 92%
time	3.17 seconds
time per comparison	0.02 seconds

Regarding language identification, 12 files were correctly identified as being written in Portuguese and 16 as being written in English. This translates into an accuracy of 82 percent (however, the 6 files wrongly identified consisted of frame pages, which means they are discarded later).

With these 34 files, 561 comparisons are possible, but after language identification just 192 are meaningful. The 192 candidate pairs were evaluated by hand, determining that 12 of them were parallel translations.

The program identified 13 pairs of files and was correct for 11 of those 13 file pairs, a precision of 85 percent. 11 out of 12 true translation pairs were identified, a recall of 92 percent.

It took 3.17 seconds to perform the entire operation, which gives about 0.02 seconds per comparison[9].

We can estimate that the program would take about one minute to evaluate 3000 candidate pairs.

5.2 Second experiment — URLs mining

For the second experiment, a robot retrieved a list of URLs in the Portuguese domain, resulting in a list with over 800 000 entries.

We then parsed that list of links with the first approach described in section 2.1. The result was a list of blocks (2376) from which we selected the pairs with possible Portuguese and English files (756) using language standard tags. Those file pairs were then retrieved from the web and compared.

[9]these tests took place in a Pentium IV 1.5 Ghz with 256 megabytes of real memory, under Linux.

number of URLs found	850 406
number of blocks created	2 376
number of candidate pairs (PT-EN)	756
pairs successfully retrieved	496
pairs identified as true translations	253
time taken for blocks detection	1.01 minute

We should note that these URLs were grabbed without looking to it's contents. In the same point of view, it is true that this process is a lot faster to find possible candidate pairs.

6 Conclusions

Mining the web can be useful for many purposes. Using only the search engines in the web without any other tool, we can get many interesting information.

There are simple processes to check if a pair of files is a translation. In fact, after detecting each one's language and calculating similarities, there is a good of probability to know whether or not the files are translations of each other. So, if we have a small amount of pages, we can make a combinatorial comparison to detect all translations in various languages. This can be a chaos when trying to parallelize a lot of files.

Other simple processes, like the comparison of file and directory names on a well organized web site, can give good answers in almost no time. The problem is that we never know when the web site is correctly organized.

After obtaining these file pairs, and since people translate web pages instead of rewriting them in various languages, there is a good chance of aligning them immediately without many complications. On the other hand, the HTML structure for translated pages are, almost every time, the same for each translation.

Then, we can make a corpora of the aligned files, and make them accessible via web for any purpose we should need.

6.1 Future work

The following are some areas for future investigation.

- `Further Evaluation` – Though some preliminary tests have been made, the whole program must be tested with other language pairs and on larger candidate sets. The various approaches for candidate retrieval must also be tested.
- `Independent Evaluation` – We would like to follow the approach of Resnik (Resnik, 1999), having at least two human judges, deciding whether two pages are translations of each other.
- `Multilingual Support` – Although the project was intended for bilingual retrieval of corpora, many things in it, were created considering the possibility of extension to the multilingual case.

References

Almeida, J. João and Ulisses Pinto. 1994. Jspell — um módulo para análise léxica genérica de linguagem natural. In *Actas do Congresso da Associação Portuguesa de Linguística*.

Déjà vu. 1993-2002. Déjà vu. Computer Assisted Translation System, http://www.atril.com/.

Grefenstette, Gregory. 1995. Comparing two language identification schemes. In *JADT 1995, 3rd International Conference on Statistical Analysis of Textual Data*.

Hietaniemi, Jarkko. 2001. *String::Approx - Perl extension for approximate matching*.

IMS Corpus Workbench. 1994-2002. http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/.

ISO 639. 1992. *Language Codes*. International Organization for Standardization.

Niksic, Hrvoje. 2001. *GNU Wget Manual*.

Resnik, Philip. 1998. Parallel strands: A preliminary investigation into mining the web for bilingual text. In *D. Farwell, L. Gerber, and E. Hovy (eds.), Machine Translation and the Information Soup (AMTA-98)*. Lecture Notes in Artificial Intelligence 1529, Springer.

Resnik, Philip. 1999. Mining the web for bilingual text. In *37th Annual Meeting of the ACL'99*. College Park, Maryland.

Simões, Alberto and J. João Almeida. 2001. jspell.pm — um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas da Associação Portuguesa de Linguística*.

Trados. 1998-2002. Trados — language technology for your business. Computer Assisted Translation System, http://www.trados.com/.