Bitextor, a free/open-source software to harvest translation memories from multilingual websites

Miquel Esplà-Gomis
Grup Transducens, Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant
E-03071 Alacant, Spain
miquel.espla@ua.es

Abstract

Bitextor is a free/open-source application for harvesting translation memories from multilingual websites. It downloads all the HTML files in a website, preprocesses them into a coherent format and, finally, applies a set of heuristics to select pairs of files which are candidates to contain the same text in two different languages (bitexts). From these parallel texts, translation memories are generated in TMX format using the library LibTagAligner, which uses the HTML tags and the length of text chunks to perform the alignment.

1 Introduction and motivation

Since its inception, one of the most important problems in the area of machine translation research, particulary in corpus-based machine translation, has been that of obtaining parallel multilingual corpora.

Nowadays, the Internet may be seen as a large multilingual corpus as there a large number of websites in which different pages can be found containing the same text written in different languages. These pages can be considered *bitexts* (or *parallel texts*). The question remains which is the best system to find these parallel texts and obtain an aligned parallel corpus from them.

With this question in mind, Bitextor project was born. Its aim is to develop a free/open-source application capable of finding parallel texts in multilingual websites, aligning them and generating translation memories formatted in TMX.[1]

Different studies have been performed about parallel text alignment. In Bitextor we have tried to take advantage of different parameters used in alignment for comparison. The idea is that the more reliable an alignment is (it has best results for the alignment parameters used) the more probable is that the aligned texts are parallel.

The earlier approaches to parallel text alignment were based in sentence length (Brown et al., 1991; Gale and Church, 1994). These systems provide good results between languages with a high sentence length correlation, but their weakness is that many parallel texts are not sentence-to-sentence translations and it is usual to find, in example, that a sentence in a text is written using more than one sentence in its parallel.

Other systems use structural elements of a text, independently of its content, to identify parallelisms with otherwise (i.e.: Kit et al. (2005)). These systems have the advantage that they can work with more combinations of languages and are able to tolerate imperfect translations. A very interesting approach uses HTML tag structure to align. The results with this approach are shown to be better than simple sentence-length alignment (Sanchez-Villamil et al., 2006).

In this case, the goal of Bitextor is to combine HTML tag structure and sentence length information to compare files and try to determine if they are parallel texts.

[1] http://www.lisa.org/Translation-Memory-e.34.0.html [Last visited: 30th July 2009]

2 Previous works

Nowadays, Bitextor is not the only approach around the idea of using the Internet as a multilingual corpus. Probably, one of the most closely related project in this area is WeBiText (Désilets et al., 2008). This application has the goal of creating a system to find a set of words in a parallel text from a bilingual website. The main difference is that Bitextor's aim is to generate a translation memory as big as possible, independently of its content, and WeBiText's aim is to find only a parallelism for a group of words searched in a bilingual website. So, we could conclude that, although both projects use a common concept (the use of the web as a multilingual parallel corpus) WeBiText is more oriented to use by human translators and Bitextor is aimed at creating large amounts of aligned parallel texts.

By focusing on technical issues, the most important difference between both applications is that, in WeBiText, the parallel texts are searched using language markers in URLs (Nie et al., 1999) and then, the text segment lengths are used to align the files. In Bitextor, all the website is downloaded and HTML tag structure and segment length are the elements used to determine if two files are parallel or not and to align them. Therefore, Bitextor has the advantage that is not dependent on the language markers in the URL and it can extract translation memories from more websites.

3 File content representation

Bitextor uses a system to represent file contents using a string of integers that is used later to perform the comparison. These strings consist of positive and negative integers.[2] The first step to create these sequences is to segment each file considering into two kinds of elements: HTML tags and text blocks.

Although this is explained in more detail later,[3] Bitextor uses the tag categories defined in the LibTagAligner configuration file which are established by the user and are used to group tags. Tags not grouped in any category are assigned automatically to the *irrelevant* category. These tags will not be considered by the system in the comparison process; therefore, irrelevant tags will be deleted and text blocks separated by them will be concatenated in a single block.

Once the file has been segmented, the second step is to assign a unique negative integer to each tag name. Bitextor mantains a dynamic symbol table with correspondence between tag names and negative integers.

Each text block is assigned a positive number that represents its length measured in characters.

The string of integers resulting from this process will be called the *fingerprint* of the file.

4 File comparison

The key element in Bitextor is a functionality to compare pairs of files and deduce which of them are candidates to be a parallel text. To do this, it primarily uses file fingerprints. But, before comparing file fingerprints, a set of superficial heuristics are applied. Their aim is to ease the later tasks of Bitextor, so that the application does not need to process every pair of files to compare their fingerprints. Only if a pair of files passes all these tests, the fingerprint comparison will be performed.

These heuristics are:

- *Text language comparison*: It is obvious that if two files are written in the same language, one of them cannot be a translation of the other one.[4]
- *Filename extension comparison*: It is assumed that if a file in the same website is a translation of another one, both files will usually have the same filename extension.
- *File size ratio*: This parameter is relative and is used to filter pairs of files whose size is very different. The threshold for size ratio can be defined by the user.
- *Total text length difference*: This parameter has the same function that the later one, but it measures the size of each file's plain text in characters.

4.1 Fingerprint comparison process

To perform the comparison process, an adaptation of the Levenshtein edit distance algorithm (Leven-

[2] The number 0 will be considered as a positive integer representing an empty text block.

[3] See section 5.1: *The alignment process*.

[4] Section 4.3 explains how the language of a file is detected

shtein, 1966) has been applied. The cost function C for the three different kind of operations is:

For insertions $C_i(x)$ and deletions $C_d(x)$, the cost is the same for a tag and a text block (independently of length) (x):

$$C_i(x) = 1$$

$$C_d(x) = 1$$

However, for substitutions we will have two different values, depending on tags or text blocks. If we substitute tags t_1 and t_2, the cost function $C_s(t_1, t_2)$ is:

$$C_s(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 1 & \text{if } t_1 \neq t_2 \end{cases}$$

In the case of text blocks b_1 and b_2, with a threshold for text block comparison t_b, the cost function $C_s(b_1, b_2)$ for substitution is:

$$C_s(b_1, b_2) = \begin{cases} 1 & \text{if } D(b_1, b_2) > t_b \\ 0 & \text{if } D(b_1, b_2) \leq t_b \end{cases}$$

where $D(b_1, b_2)$ is the percent difference between the length of two text blocks b_1 and b_2:

$$D(b_1, b_2) = \frac{|b_1 - b_2|}{\max(b_1, b_2)} \cdot 100\%$$

In the case of comparison between a tag and a text block, the result will be infinite.

With this system, an integer representing the distance between both fingerprints is obtained and this value will be used to discard the pairs of files when it is larger than a certain threshold.

For example, we can see the result for the algorithm with a text block comparison threshold $t_b = 20\%$ in Figure 3, where the result of the edit distance algorithm applied on the fingerprints extracted from the files in Figure 1 and Figure 2 is represented.

```
<div>
    <h1>
        Este és un exemple de títol
    </h1>
    Este és un exemple de fragment de text.
    <a href="http://www.anurl.cat/cat">
        <img src="an_image.png">
    </a>
</div>
```

Figure 1: HTML File 1 (in Catalan).

```
<div>
    <h1>
        This is an example of a title
    </h1>
    This is an example of a piece of text.
    <a href="http://www.anurl.cat/eng">
        <img src="an_image.png">
    </a>
    <br />
</div>
```

Figure 2: HTML File 2 (in English).

		-1	**-2**	**27**	**-2**	**39**	**-4**	**-5**	**-4**	**-1**
	0	1	2	3	4	5	6	7	8	9
-1	1	0	1	∞	4	∞	6	7	8	8
-2	2	1	0	∞	5	∞	7	7	8	9
27	3	∞	∞	0	∞	6	∞	∞	∞	∞
-2	4	4	5	∞	0	∞	7	8	9	10
38	5	∞	∞	6	∞	0	∞	∞	∞	∞
-4	6	6	7	∞	7	∞	0	1	2	3
-5	7	7	8	∞	8	∞	1	0	1	2
-4	8	8	8	∞	9	∞	2	1	0	1
-6	9	9	9	∞	10	∞	3	2	1	1
-1	10	10	10	∞	11	∞	4	3	2	1

Distance=1

Figure 3: Optimal path in the edit distance algorithm.

Fingerprint comparison may be seen as using two thresholds. The first one is an absolute threshold (t_a) and it will be compared directly with the result of the edit distance algorithm. If the resulting value from the edit distance is larger than this threshold, the pair of files compared will be discarded.

The second threshold is relative (t_r). It is defined by the user as a percentage over the maximum length of one of both fingerprints. This percentage can be converted into an absolute value by applying this function on the two fingerprints (f_1 and f_2):

$$t'_a = \max(\text{Len}(f_1), \text{Len}(f_2)) \cdot \frac{t_r}{100\%}$$

where $\text{Len}(f)$ is a function that calculates the length of a fingerprint.

For a pair of files, only the most restrictive threshold will be applied. The absolute one is the main threshold, but, in files with a very short fingerprint, it can be useless because all possible results may be

lower than this limit. This is the reason why the relative threshold is defined. This idea is illustrated with the Figure 4, where $t_a = 5$ and $t_r = 20\%$.

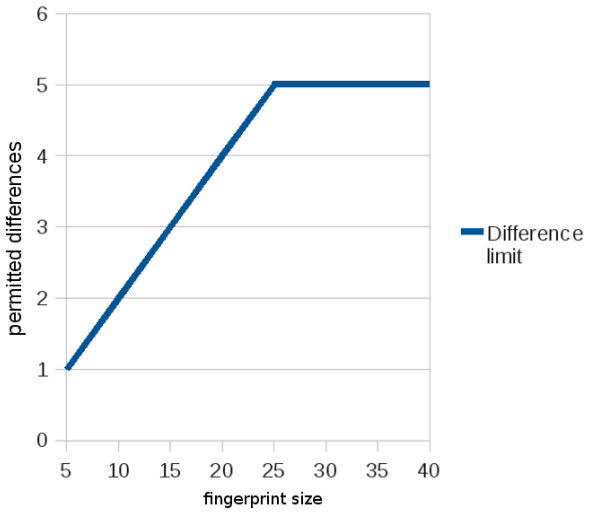

Figure 4: Relation between fingerprint size and permitted differences between them.

4.2 Optimisation of the algorithm

To improve the speed of the application, the customary *beam-search* modification to the original edit-distance algorithm has been applied, to change the complexity in the comparison of two fingerprints with lengths m and n from $O(m \cdot n)$ to $O(\max(m, n))$.

If we look at the edit distance algorithm implemented, we can see that when two fingerprints are similar, their optimal path[5] will be close to the straight diagonal line in the table. In our case, we are only interested in very similar fingerprints, and, as a consequence, all the fingerprints which do not have their optimal path in a relatively narrow beam around the diagram edit distance algorithm table can be safely discarded. Knowing this, it can be concluded that it is only necessary to calculate that diagonal beam in the algorithm table.

If we take the example in the last section and limit the calculations to a diagonal of width 2, the algorithm will make the calculations on the beam represented in the Figure 5. As can be seen, the result would be the same as in the exact calculation at a much lower cost. For long fingerprints the difference could be very significant.

		-1	-2	27	-2	39	-4	-5	-4	-1
	0	1								
-1	1	0	1							
-2	2	1	0	∞						
27		∞	∞	0	∞					
-2			5	∞	0	∞				
38				6	∞	0	∞			
-4					7	∞	0	1		
-5						∞	1	0	1	
-4							2	1	0	1
-6								2	1	1
-1									2	1

Figure 5: Optimal path in edit distance algorithm.

A very important question is the relation between the fingerprint comparison threshold (t_a and t_r) and the width of the beam is that, if the beam is too narrow, the optimal path may lie out of it. This would mean that the final result of the algorithm would not be the optimal and we may discard some pairs of files that should not be discarded. Then, it is important to adapt the beam width to fingerprint thresholds.

5 Steps in Bitextor's processing

5.1 Downloading

The downloading phase is performed by using the application HTTrack.[6] The application downloads all the HTML files from the multilingual website. Downloading respects the directory tree structure. It is necessary to emphasise the importance of downloading the whole website before starting the comparison process. During the comparison we will need to validate every file with the others and we will need to have all the set of files available. This system can be limited to neighbouring levels in the directory tree. Because of this, in the future, Bitextor will be able to download every level of the directory tree only when it is used.

[5] The path with the lowest cost from the upper-left corner to the bottom-right corner in the algorithm table. Such as the shaded path in Figure 3.

[6] `http://www.httrack.com` [Last visited: 30th July 2009]

5.2 Preprocessing and information harvesting

During this phase, all the downloaded files will be preprocessed to adapt them to the necessary requirements for subsequent phases. For this reason, Bitextor uses the LibTidy[7] library to standardise probably invalid HTML files into valid XHTML files. With this, we can guarantee that the structure of tags is correct. The original character encoding is also converted to UTF-8.[8]

Once a file has been preprocessed, the next step is to collect some information necessary to compare the files and generate the translation memories, such as the name and filename extension of the files. The language in which each text has been written is also detected and saved by using LibTextCat.[9] This library detects the language in which a text has been written using an n-gram model extracted from a corpus of texts written in this language. N-gram models for new languages can be added using the configuration file.

The file's fingerprint is also obtained in this step.

Finally, information obtained from the files is saved in a list organised according to the depth in the directory tree where the analysed file has been found. Organising information in this way makes access easier since comparison between files is performed level by level.

5.3 Comparison between files and translation memory generation

During this phase, the file comparison is performed. The process is based on level comparisons. As mentioned, the user can limit the depth difference in the directory tree when comparing. Parallel texts are usually in the same level, or in very close levels. As a result of this, it is not necessary to compare each file with all the others. We will only need to compare files within the defined level interval.

Then, to optimise the process, Bitextor works in the following way: first, it loads all the levels allowed by the level of comparison limitation. Then, it gets the first file in the list of the upper level loaded and compares it with all the other files (including files in its same level). Once it has been compared, this file is discarded and another one from its same level is loaded. This process is repeated until all files in the upper level loaded have been removed. Then, a new level will be loaded and the process will start again. Finally, when there are not more levels to load, the remaining files will be compared between them.

A diagram that explains the process can be seen in Figure 6.

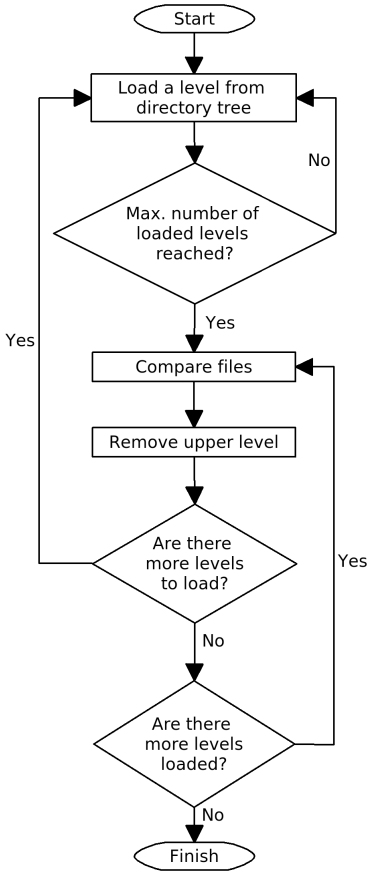

Figure 6: File comparison process.

6 Generation of translation memories with LibTagAligner

Bitextor uses the LibTagAligner library to generate translation memories in TMX format.

[7] `http://tidy.sourceforge.net` [Last visited: 30th July 2009]

[8] To detect character encoding, Bitextor uses the LibEnca library: `http://sourceforge.net/projects/freshmeat_enca/` [Last visited: 30th July 2009]

[9] `http://software.wise-guys.nl/libtextcat/` [Last visited: 30th July 2009]

6.1 The alignment process

The system employed by LibTagAligner to align two HTML files is based on a very similar method to the one used by Bitextor to compare files. The main difference between them is that LibTagAligner uses a finer representation to compare the files. As in Bitextor's fingerprint system, TagAligner uses integers to represent tags and text blocks. But, in addition, TagAligner uses tag categories. These categories are defined by the user and group tags of the same kind. The objective of grouping tags is to allow assigning different weights to every possible edit-distance algorithm operation between tag categories. This system obtains a higher precision in the alignment process than text-length-based ones (Sanchez-Villamil et al., 2006).

Tag categories are defined through a configuration file where the user defines their names and the list of tags contained in each category. There is only one predefined kind of category: the *irrelevant* category. Here is where all the uncategorised tags are collected to be discarded during the alignment process.

For a tag t, the cost of an insertion $C_i(t)$ or deletion $C_d(t)$ operation will be expressed by the functions:

$$C_i(t) = W_i(t)$$
$$C_d(t) = W_d(t)$$

where $W_i(t)$ and $W_d(t)$ are the functions that return the weights assigned by the user for the insertion and deletion operations.

In the case of substitution, the cost function on two tags (t_1 and t_2) will be:

$$C_s(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ W_s(t_1, t_2) & \text{if } t_1 \neq t_2 \end{cases}$$

where $W_s(t_1, t_2)$ is the function that determines the cost of a substitution on a pair of tags.[10]

Weights are also assigned to text block operations. The main difference with the operation on tags is that in the case of the text blocks the cost is not constant, but depends instead on the difference between their lengths. Then, in our case, the set of cost functions for text blocks b is:

$$C_i(b) = W_i(b) \cdot \mathrm{Len}(b)$$

[10]Those values can be found in Sanchez-Villamil et al. (2006).

URL	Possible correct pairs	Generated pairs	Correct generated pairs
http://www.fpc.upc.edu	22	13	13
http://congreso.lliurex.net	89	74	59

Table 1: Results obtained with Bitextor on the websites.

$$C_d(b) = W_d(b) \cdot \mathrm{Len}(b)$$

$$C_s(b_1, b_2) = W_s(b_1, b_2) \cdot |\mathrm{Len}(b_1) - \mathrm{Len}(b_2)|$$

Obviously, tag-text block substitutions are not permitted. To that effect, the cost of the operation will be infinite.

6.2 Preliminary results

To perform a preliminary evaluation of the first results obtained by the application, a pair of tests have been performed to get a basic idea of Bitextor's current potential. A small pair of trilingual websites (Catalan–Spanish–English) have been selected to perform the test. In this case, the size of the websites is important to allow full evaluation: we must choose websites that are not too large since we must be able to do the file comparison process by hand to compare them with the results obtained by Bitextor.

The objective of this test is to obtain information about the precision and recall achieved by Bitextor. Precision is the proportion of files paired up correctly from the total of files paired up. Recall is the proportion of files paired up correctly over the number of total files that could have been paired up correctly.

We have focused these tests on precision and not on recall. To do this, the fingerprint comparison threshold has been set to 5 (this means that fingerprints can only have five differences between them). This is because it seems reasonable for a Bitextor user to require quality in results and not quantity, to make later correction easier.

The results of the test are presented in Table 1. The first website has been processed (without counting downloading time) in 3.5 seconds and the second one, has been processed in 13.7 seconds. Precision and recall percentages are given in Table 2.

As can be seen, there is room for improvement. In any case, the precision has been 100% in both

URL	Recall	Precision
http://www.fpc.upc.edu	63%	100%
http://congreso.lliurex.net	83%	100%

Table 2: Recall and precision results.

cases.[11]

7 Conclusions and future work

At the moment, Bitextor produces promising results. It carries out with a reasonable efficiency the objectives raised at the starting of the project: it is able to generate translation memories automatically from multilingual websites with a reasonable level of precision. In fact, its high configurability allows the users to optimise the results to the balance precision-recall.

However, the results could be improved to a large degree, mainly, regarding recall. One of the key elements to do this is LibTagAligner's aligning algorithm. It would be very interesting to take advantage of this algorithm's precision in comparison process without increasing dramatically the running time of Bitextor.

It would also be important to increase the integration between Bitextor and the application HTTrack to optimise the downloading process to only save on disk the absolutely necessary files. It would allow Bitextor to be used in computers with a smaller hard disk.

We plan to work with other free/open-source projects (i.e.: bitext2tmx)[12] to integrate Bitextor (and TagAligner) into them. With this, it could be possible to build a bigger framework for bitext harvesting and correcting to help users to generate their own multilingual corpora.

A final objective for the future in Bitextor project is the generation and publication of corpus of TMX translation memories harvested from multilingual websites.

[11] Tag and sentence alignment parameters have been taken from Sanchez-Villamil et al. (2006). In this article results for sentence alignment are also present.

[12] `http://www.sourceforge.net/projects/bitext2tmx` [Last visited: 30th July 2009]

8 About Bitextor

Bitextor and LibTagAligner are free/open-source applications released under the General Public License version 2.0 (GPL v2). [13] They are available for UNIX-based platforms. Code and releases can be found at `http://sourceforge.net/projects/bitextor` and `http://sourceforge.net/projects/tag-aligner`.

9 Acknowledgements

The original development of Bitextor was funded by the Ministerio de Ciencia y Tecnología (Spanish Government) between 2004 and 2006 through grant (TIC2003-08681-C02).

The Bitextor Project is currently funded by the Universitat d'Alacant.

Enrique Sánchez Villamil was the author of the initial version (v1.0) of TagAligner (on which the initial version of LibTagAligner, used by Bitextor was based).

Miquel Simón i Martínez was the author of the second version (v2.0) of TagAligner, with an improvement in configurability through the incorporation of an XML configuration file.

Many thanks to Francis M. Tyers and Mikel L. Forcada for comments on the manuscript.

References

Brown, P., Lai, J., and Mercer, R. (1991). Aligning sentences in parallel corpora. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 169–176. Association for Computational Linguistics Morristown, NJ, USA.

Désilets, A., Farley, B., Stojanovic, M., and Patenaude, G. (2008). WeBiText: Building Large Heterogeneous Translation Memories from Parallel Web Content. *Proc. of Translating and the Computer*, 30:27–28.

Gale, W. and Church, K. (1994). A program for aligning sentences in bilingual corpora. *Computational linguistics*, 19(1):75–102.

Kit, C., Liu, X., Sin, K., and Webster, J. J. (2005). Harvesting the bitexts of the laws of Hong Kong from the Web.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710.

[13] `http://www.gnu.org/licenses/gpl-2.0.html` [Last visited: 30th July 2009]

Nie, J., Simard, M., Isabelle, P., and Durand, R. (1999). Cross-Language Information Retrieval based on Parallel Texts and Automatic Mining of Parallel Texts from the Web. In *Proceedings of SIGIR'99: 22nd International Conference on Research and Development in Information Retrieval: University of California, Berkeley, August 1999*, page 74. Association for Computing Machinery (ACM).

Sanchez-Villamil, E., Santos-Anton, S., Ortiz-Rojas, S., and Forcada, M. (2006). Evaluation of alignment methods for HTML parallel text. *Lecture Notes in Computer Science*, 4139:280.