A Two-Level Structure for Compressing Aligned Bitexts[⋆]

Joaquín Adiego,[†] Nieves R. Brisaboa,[‡] Miguel A. Martínez-Prieto,[†]
and Felipe Sánchez-Martínez[♮]

[†]Dept. de Informática, Universidad de Valladolid, Spain.
{jadiego|migumar2}@infor.uva.es
[‡]Database Lab, Universidade da Coruña, Spain.
brisaboa@udc.es
[♮]Dept. de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, Spain.
fsanchez@dlsi.ua.es

Abstract. A *bitext*, or *bilingual parallel corpus*, consists of two texts, each one in a different language, that are mutual translations. Bitexts are very useful in linguistic engineering because they are used as source of knowledge for different purposes. In this paper we propose a strategy to efficiently compress and use bitexts, saving, not only space, but also processing time when exploiting them. Our strategy is based on a two-level structure for the vocabularies, and on the use of *biwords*, a pair of associated words, one from each language, as basic symbols to be encoded with an ETDC [2] compressor. The resulting compressed bitext needs around 20% of the space and allows more efficient implementations of the different types of searches and operations that linguistic engineerings need to perform on them. In this paper we discuss and provide results for compression, decompression, different types of searches, and bilingual snippets extraction.

1 Introduction

The amount of multilingual texts is growing very fast due to multilingual digital libraries and legal requirements in countries and supra-national entities with more than one official language. Two texts that are mutual translations are usually referred to as a *bilingual parallel corpus* or, in short, as a *bitext*. The growing availability of bitexts has enabled the development on many natural language processing applications that use bitexts as source of knowledge.

Usually, bitexts get *aligned* before exploiting them; a standard *text alignment* process allows to establish word correspondences between the two texts of the bitext. Aligned bitexts can be used in applications involving both languages (machine translation, cross-language information retrieval, extraction of bilingual lexicons, etc) or in monolingual applications (syntactic parsing, word sense induction, word sense disambiguation, etc.) that use the bitexts as a bridge to project the linguistic knowledge available in one language to another one [11].

[⋆] Funded by Spanish projects TIN2006-15071-C03-01, TIN2006-15071-C03-02 and TIN2006-15071-C03-03. The work of Miguel A. Martínez-Prieto is supported by a fellowship granted by the Regional Government of Castilla y León and the European Social Fund.

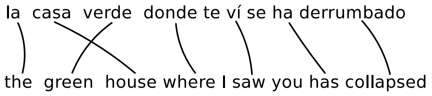
la casa verde donde te ví se ha derrumbado
the green house where I saw you has collapsed

Fig. 1. Spanish–English word-aligned sentence.

We present a strategy to compress bitexts that we called *Two-level Compressor for Aligned Bitexts* (2LCAB). Our strategy is designed to facilitate the use of the most interesting features of bitexts, because, in our compressed representation, obtaining the words in one language aligned with a word in the other language is simply done by using a vocabulary, instead of processing the whole aligned bitext. In addition, 2LCAB obtains compression ratios around 20% and allows a more efficient processing of the aligned bitexts that the uncompressed form.

2 Word-Aligned Bitexts

A *bitext* is a text written in two languages. In words of Melamed, "bitexts are one of the richest sources of linguistic knowledge because the translation of a text into another language can be viewed as a detailed annotation of what that text means" [10].

A bitext in which the translation relationship among the words in one text (left) and the words in the other text (right) has been established is usually referred to as a *word-aligned* bitext; the task of establishing such relationships is known as *word alignment*.

The *word alignment* task [15] connects words in the left sentence L with words in the right sentence R. The result is a bigraph for the words in L and the words in R with an arc between word $l \in L$ and word $r \in R$ if and only if they are mutual translations. Figure 1 shows an example of a Spanish–English word-aligned sentence.

For this research the bigraph representing a word-aligned bitext is stored as a sequence of pairs of two words, each one from a different language, that are mutual translations in the bitext. Therefore, for this research the word-aligned bitext of the example in Figure 1 is represented as the following sequence of pairs:

`(la,the) (,green) (casa,house) (verde,) (donde,where) (te,) (,I) (ví,saw) (,you) (se,) (ha,has) (derrumbado,collapsed)`

Notice that some words are associated to an "empty word", e.g. `(te,)`. This is either because that word is not aligned with another word in the other text, or because its alignment has been discarded due to a crossing, e.g. `(,green)`. In this work we have used one-to-one word alignments obtained with the help of the open-source GIZA++ [15] toolkit.[1]

3 Compression of Natural-Language Texts

The key to the success of natural language text compression is the use of a *word-based* model, so that the text is regarded as a sequence of words. This poses the overhead

[1] http://code.google.com/p/giza-pp/

of managing a large source alphabet, but in large text collections the vocabulary size is relatively insignificant because of Heaps Law [5]. In order to be searchable, semi-static models have been used in compressed text databases, to ensure that the codeword assigned to a word does not change across the text. Thus, a pattern can be compressed and directly searched for in the compressed text without decompressing it. This is also essential to allow local decompression of text passages in order to present them to the final users.

End-Tagged Dense Code (ETDC) [2] is a *word-based* compression technique where the first bit of each byte is reserved to flag whether the byte is the last one of its codeword (*stopper*) or not (*continuer*); this flag is enough to ensure that the code is a prefix code regardless of the content of the other 7. The flag bit in ETDC permits Boyer-Moore-type searching [1] and random access. Simple encode and decode procedures can be used to obtain the codeword C_i corresponding to a position i in the sorted vocabulary ($C_i = \text{encode}(i)$) and, symmetrically, to obtain the position i corresponding to a specific codeword C_i ($i = \text{decode}(C_i)$).

3.1 Compression of Bitexts

Compression of bitexts is a subfield of natural language text compression. In spite of its relevance, only few previous works have been found in the literature. In [14] text compression methods are considered for its extension to bitext compression considering exact correspondences between two words, and synonymy relationships between the words in both texts (as given by a thesaurus). These parallel predictions are then combined with PPM [3] ones. The weighting of both models are carefully tuned improving PPM compression ratios on separate texts.

Text alignment is proposed in [4] as a way to enable multilingual text compression. The algorithm stores one of the texts (L) as it is, and the other one (R) as a collection of pointers to the translation of the substring in the L text. These relationships are determined by means of an alignment algorithm that uses some additional linguistic resources, such as a lemmata dictionary in L and a bilingual glossary, among others.

4 Two-Level Compressor for Aligned Bitexts (2LCAB)

Our strategy, called *Two-Level Compressor for Aligned Bitexts (*2LCAB*)*, is based on two main ideas: (i) the use of *biwords* [9], pairs of aligned words, as the basis of the model, that is, as the symbols to compress, and (ii) the use of a two level structure for the representation of the vocabularies, where the vocabulary of biwords, at the second level, is represented in compressed form using the vocabularies of the first level.

Figure 2 shows a conceptual description of this scheme. At the first level two vocabularies are stored, one for each language. Each of them stores the words of the corresponding language sorted by the number of biwords they take part in. The “empty word” is also represented in both dictionaries. On the second level, each pair of words (biword) is represented as the concatenation of the codewords assigned to each word in the pair using ETDC. That is, each biword is used as a single symbol in the *biword vocabulary*. In this second-level vocabulary the ranking of biwords is performed in accordance with their frequencies in the bitext.

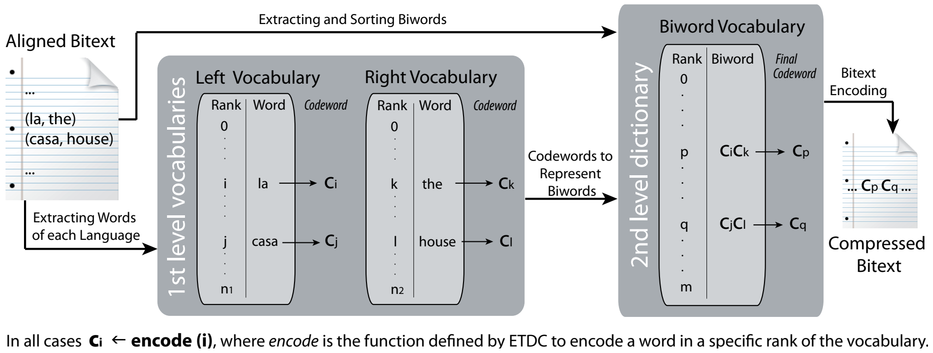

Fig. 2. Conceptual description of the 2LCAB strategy.

Four strings are the output of the compression process. Lv and Rv contain sorted left and right vocabularies. BWv stores the biword vocabulary where biwords are represented in compressed form as explained above. These three strings constitute the header of the compressed bitext. The fourth string contains the compressed bitext, where each pair of words is represented by the codeword corresponding to the ETDC codeword assigned to its biword from its position in the second level vocabulary.

4.1 Compression and Decompression

Our strategy is based on a semi-static approach; therefore, it is necessary to make two passes over the (aligned) bitext. In the first one, the aligned bitext is pair-to-pair parsed and, in addition to the three vocabularies aforementioned, a hash table of pairs (biword, codeword) is built. In the second pass, the compression process looks for each biword in the hash table and outputs its corresponding codeword. The compression process is completed in $O(n)$ time overall, where n is the number of biwords in the bitext.

The decompression process begins by loading the strings Lv and Rv to get the left and right vocabularies. These strings are stored in vectors V_l and V_r, respectively. Then, the string BWv is read and V_l and V_r are used to rebuilt the biword vocabulary, which is stored in vector V_b, where each biword is explicitly represented by its pair of words so as to improve the efficiency of the decompression process. Building V_b takes $O(b)$ time, where b is the number of entries in the BWv vocabulary. Then, the compressed bitext is processed by decoding each codeword. Given a codeword C_i, the simple decoding function of the ETDC is used to obtain the corresponding position $i = \mathrm{decode}(C_i)$ in the biword vocabulary ($V_b[i]$). The decompression process is completed in $O(T)$ time, where T is the number of biwords in the bitext.

4.2 Processing the Compressed Bitext

Our representation allows to process the bitext without decompressing it. In fact, only decompressing small snippets is necessary for most applications, and only when they need to show the snippet to the user. Semantic relationships between languages in the bitext suggest specific search possibilities such as: (i) to find all the occurrences of a word in the bitext, that is, all the occurrences of biwords that include it; and (ii) to find all the possible translations of a word, that is, all the biwords for a specific word.

To process the compressed bitext we start loading and storing the strings `Lv` and `Rv` into V_l and V_r vectors, respectively. Also, two hash tables are built from `Lv` and `Rv`. Then, the string `BWv` is read and stored in main memory. To facilitate the searches, a *bitmap* with a bit for each byte in `BWv` is built. In this bitmap 0-bits correspond to *continuer* bytes in `BWv` whereas 1-bits correspond to *stopper* bytes.

Searching the Occurrences of a Word in the Bitext. This operation is useful to retrieve all the contexts (snippets) in which each biword appears, that is, to find all the occurrences of a specific biword and decode its snippets. Given a word, and the language in which it is represented, we first find it in the corresponding first-level vocabulary (left or right hash table depending of the language supplied); this process takes $O(1)$ time. Once the codeword is retrieved it is searched in `BWv` to find those biwords in which the word appears. This is carried out using any well-known exact pattern matching algorithm (such as KMP [7] or BM [1]) slightly modified to avoid possible false matchings due to the fact that ETDC codes are not suffix codes, and, therefore, a codeword can be a suffix of another one. This overhead in searches is negligible because checking the previous byte is only necessary when a matching occurs, which is infrequent [2].

To determine the language of a codeword found in the biwords vocabulary at position p a $\mathrm{rank}_1(p)$ operation on the bitmap is done. If an *even* value is obtained, the word belongs to the left vocabulary, whereas an *odd* value means that the word belongs to the right vocabulary. If the found matching corresponds to the adequate language, the codeword of the biword is computed as $C = \mathrm{encode}(\mathrm{rank}_1(p)/2)$. Then, that codeword is added to the trie of searched codewords that will be used by the multiple-pattern matching algorithm over the compressed bitext. The search of all the required biwords takes $O(b)$ time, where b is the size of `BWv` because the rank operation, to check the language correspondence, only takes $O(1)$ [12]. At the end of this process the codewords in the trie will encode all the biwords where the searched word appears. We choose Set Horspool [6,13], as search algorithm because it is an efficient choice for very small sets of searched patterns on large alphabets. Set Horspool outputs all the occurrences of the required biwords in the compressed text. This search takes $O(m)$ time, where m is the size in bytes of the compressed text.

To find the context where each specific translation (biword) of a word is found, it is only necessary to decompress a snippet around each occurrence. Doing this is straightforward by using the ETDC decode procedure.

Searching All the Possible Translations of a Word. This operation allows to find all the correspondences of a word in a language with words in the other language. One of the main advantages of our approach is that to find all the possible translations of a word in the bitext it is not necessary to read the bitext, because all the biwords (possible translations) are represented in the vocabulary of biwords. Therefore, it is only necessary to search for the codeword of that specific word in the `BWv` string using the strategy already explained.

5 Experimental Evaluation

All the experiments were performed on a Debian 4 Etch operating system, running on an AMD Athlon Dual Core processor at 2 GHz and with 2 GB of RAM. We used g++ 4.1.2 compiler with full optimization. We used heterogeneous corpora with different languages pairs to evaluate the influence of the similarity between the two languages of the bitext in the compression ratio. Furthermore, we used bitexts of different size for each language pair. More precisely, we used bitexts of around 1, 10, and 100 MB where larger bitexts contained the smaller ones (for Spanish–Galician we only used bitexts of 1, and 10 MB). The following corpora were used:

- a Spanish–Catalan (es-ca) bitext from *El Periódico de Catalunya*,[2] a daily newspaper published both in Catalan and Spanish;
- a Spanish–Galician (es-gl) bitext from *Diario Oficial de Galicia*,[3] the bulletin of the Government of Galicia, published both in Galician and Spanish; and
- bitexts for German–English (de-en), Spanish–English (es-en) and French–English (fr-en) from the *European Parliament Proceedings Parallel Corpus* [8].

To evaluate the success of 2LCAB in obtaining a competitive compression ratio, we compare it with some well-known state-of-the-art compressors such as GZIP, BZIP2 and PPMDI [16], this last one as a representative PPM [3]. Moreover, to evaluate the effect of our strategy of using a biword-oriented model, we also implemented ETDC compression over the bitext using two different word-oriented models. In one case (1V) we just used one vocabulary to store the words of both languages. In the other case (2V) we used two different vocabularies, one for each language.

Table 1 summarizes some data about the bitexts and the compression ratios obtained by the different compressors. Notice that 2LCAB achieves very good compression ratios (some times the best one) when the size of the bitext is medium or large. However, GZIP, BZIP2, and PPM, not being semi-static, provide better results for small files. 2LCAB outperforms GZIP for 10 MB bitexts (except for es-gl which is a special case because Spanish and Galician are closely-related languages) and only PPM, as would be expected, can compete with 2LCAB for large bitexts. Nevertheless, GZIP, BZIP2, and PPM, as dynamic compressors, do not permit random access, nor direct searching.

We do not compare our compressor against those described in Section 3.1 because we have not found any available implementation. However, Conley and Klein [4] compare their TRANS approach with GZIP and BZIP2 and they conclude that TRANS is slightly better than BZIP2 (an improvement of 1% is reported). In any case, the authors do not consider the size of the auxiliary files that TRANS requires to decompress the bitext; thus, TRANS compression ratio would be worse than that of BZIP2.

Table 2 shows compression and decompression times (in seconds) for two bitext collections: es-ca and es-en. Similar times were obtained for the remaining bitexts. The times reported correspond to the average time obtained for 5 different executions. 2LCAB is always the fastest in compression, around 3-6 times faster than BZIP2 and

[2] http://www.elperiodico.com

[3] http://www.xunta.es/diario-oficial

BITEXT	SIZE (MB)	Words Left	Words Right	Biwords	GZIP	BZIP2	PPM	1V	2V	2LCAB
es-gl	1.09	6488	6543	7219	17.09%	11.21%	8.72%	32.09%	35.30%	25.35%
	10.55	24983	25284	29855	16.91%	10.58%	8.68%	28.67%	29.82%	18.53%
es-ca	1.18	15594	14939	19336	31.48%	23.09%	20.75%	47.89%	50.02%	42.78%
	11.53	54115	52256	78825	31.13%	22.18%	20.33%	36.68%	37.21%	26.72%
	105.36	161132	159216	292994	30.79%	21.95%	20.17%	32.75%	32.51%	19.97%
es-en	1.08	9169	6696	21301	31.83%	22.33%	20.07%	43.47%	42.85%	37.30%
	10.91	30486	19465	93544	31.67%	21.73%	19.98%	35.61%	34.54%	25.99%
	110.60	81868	51353	347866	31.26%	21.22%	19.48%	32.36%	31.19%	20.86%
fr-en	1.08	8211	6493	20491	31.64%	21.99%	19.78%	42.30%	41.97%	36.33%
	10.74	25536	19045	86353	31.43%	21.42%	19.74%	35.10%	34.11%	25.55%
	109.45	65877	50418	322618	31.26%	21.22%	19.52%	32.40%	31.21%	21.04%
de-en	1.08	9957	6514	21159	32.46%	22.76%	20.66%	44.69%	44.09%	39.16%
	10.94	39287	19305	90815	32.27%	22.13%	20.47%	36.38%	35.31%	27.44%
	110.86	139012	51018	357753	32.22%	22.05%	20.37%	32.75%	31.57%	22.01%

Table 1. Compression ratios.

	es-ca								es-en							
SIZE	*Compression time (secs.)*				*Decompression time (secs.)*				*Compression time (secs.)*				*Decompression time (secs.)*			
(MB)	GZIP	BZIP2	PPM	2LCAB	GZIP	BZIP2	PPM	2LCAB	GZIP	BZIP2	PPM	2LCAB	GZIP	BZIP2	PPM	2LCAB
1	0.24	0.41	0.51	0.35	0.04	0.14	0.55	0.05	0.23	0.38	0.47	0.25	0.03	0.13	0.52	0.04
10	1.32	4.16	4.61	1.10	0.16	1.40	5.15	0.33	1.36	4.10	4.70	1.01	0.14	1.35	5.16	0.29
100	10.19	38.32	41.86	6.88	1.38	12.47	45.34	1.29	11.56	41.21	45.95	7.81	1.46	13.56	50.27	1.46

Table 2. Compression and decompression times.

	Biwords	Occurrences	1V		2V		2LCAB	
			$\overline{time}$	σ	$\overline{time}$	σ	$\overline{time}$	σ
[A]	41.20	754.07	0.635	0.034	0.617	0.036	0.119 (*0.006*)	0.030 (*0.002*)
[B]	14.30	226.77	0.636	0.009	0.613	0.014	0.099 (*0.008*)	0.044 (*0.003*)
[C]	4.77	69.87	0.641	0.063	0.614	0.045	0.066 (*0.007*)	0.012 (*0.002*)
[D]	1.47	10.87	0.631	0.017	0.615	0.018	0.061 (*0.007*)	0.011 (*0.002*)

Table 3. Searching times. The values between brackets show the average time and standard deviation needed to locate all the biwords in which a given word occurs.

PPM for medium-large file sizes. Only for small files GZIP shows a slightly better performance. When decompressing, only GZIP is slightly faster than our approach, which is much faster than BZIP2 (up to 9 times) and PPM (up to 35 times).

Table 3 shows the time required to retrieve all the occurrences of a specific word when searching the es-en bitext of 100 MB. Considering the number of biwords a words is associated to, we defined four categories: [A]: $x_A \geq 25$; [B]: $7 \leq x_B \leq 24$; [C]: $3 \leq x_C \leq 6$; and [D]: $x_D \leq 2$, where $x_{category}$ indicates the number of biwords a word must be associated to, in order to belong to that category. Then, we built four groups of 30 words randomly chosen among those in each category.

We used 1V and 2V to compare the efficiency of 2LCAB in searching processes. Notice that, 1V performs a simple pattern-matching process to find the codeword of the desired word. However, 2V needs to check if the found codeword belongs to the appropriate side of the bitexts, that is, if it is in the desired language; this is achieved by calculating if the found codeword is in a even or an odd position.

2LCAB is the fastest choice in all the cases, improving 5-10 times both 1V and 2V compressed bitexts. Notice that the number of occurrences has a stronger influence in 2LCAB than in 1V and 2V. This is because when there are many biwords associated to a word the Set Horspool algorithm handles a more complex trie composed by all the codewords of those biwords. Finally, notice that 2V is always slightly better than 1V. This is mainly due to the fact that 2V gets better compression ratios than 1V.

6 Conclusions

2LCAB has been proposed as strategy to compress word-aligned bitexts. It provides very good compression ratios and it is the fastest option for compressing and decompressing large bitext. Its main property is that bitexts can be efficiently exploited because different kind of searches and local decompression can be effectively performed over the compressed bitext without needing to decompress it. Another interesting result of this research is how the similarity between the languages in the bitext affects the number of different biwords, the number of total biwords and, therefore, the compression ratio.

References

1. R.S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. of ACM*, 20(10):762–772, 1977.
2. N. R. Brisaboa, A. Fariña, G. Navarro, and J. R. Paramá. Lightweight natural language text compression. *Information Retrieval*, 10(1):1–33, 2007.
3. J. G. Cleary and I. H. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Trans. on Communications*, COM-32(4):396–402, April 1984.
4. E. S. Conley and S. T. Klein. Using alignment for multilingual text compression. *Intl. J. of Foundations of Computer Science*, 19(1):89–101, 2008.
5. H.S. Heaps. *Inf. Retrieval - Computational and Theoretical Aspects*. Academic Press, 1978.
6. R.N. Horspool. Practical fast searching in strings. *Softw. Pract. & Exper.*, 10:501–506, 1980.
7. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. of Computing*, 6(2):323–350, 1977.
8. P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proc. of the* 10^{th} *Machine Translation Summit*, pages 79–86, 2005. http://www.statmt.org/europarl/.
9. M. A. Martínez-Prieto, J. Adiego, F. Sánchez-Martínez, P. de la Fuente, and R. C. Carrasco. On the use of word alignments to enhance bitext compression. In *Data Compres. Conf.*, page 459, 2009.
10. I. D. Melamed. *Emplirical methods for exploting parallel texts*. MIT Press, 2001.
11. R. Mihalcea and M. Simard. Parallel texts. *Natural Language Eng.*, 11(3):239–246, 2005.
12. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comp. Surv.*, 39(1), 2007.
13. G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.
14. C. G. Nevill-Manning and T. C. Bell. Compression of parallel texts. *Information Processing & Management*, 28(6):781–794, 1992.
15. F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Comp. Linguistics*, 29(1):19–51, 2003.
16. D. Shkarin. PPM: One Step to Practicality. In *Data Compres. Conf.*, pages 202–211, 2002.