Sub-sentential Alignment Using Substring Co-Occurrence Counts

Fabien Cromieres
GETA-CLIPS-IMAG
BP53 38041 Grenoble Cedex 9
France
fabien.cromieres@gmail.com

Abstract

In this paper, we will present an efficient method to compute the co-occurrence counts of any pair of substring in a parallel corpus, and an algorithm that make use of these counts to create sub-sentential alignments on such a corpus. This algorithm has the advantage of being as general as possible regarding the segmentation of text.

1 Introduction

An interesting and important problem in the Statistical Machine Translation (SMT) domain is the creation of sub-sentential alignment in a parallel corpus (a bilingual corpus already aligned at the sentence level). These alignments can later be used to, for example, train SMT systems or extract bilingual lexicons.

Many algorithms have already been proposed for sub-sentential alignment. Some of them focus on word-to-word alignment ((Brown,97) or (Melamed,97)). Others allow the generation of phrase-level alignments, such as (Och et al., 1999), (Marcu and Wong, 2002) or (Zhang, Vogel, Waibel, 2003). However, with the exception of Marcu and Wong, these phrase-level alignment algorithms still place their analyses at the word level; whether by first creating a word-to-word alignment or by computing correlation coefficients between pairs of individual words.

This is, in our opinion, a limitation of these algorithms; mainly because it makes them rely heavily on our capacity to segment a sentence in words. And defining what a word is is not as easy as it might seem. In peculiar, in many Asians writings systems (Japanese, Chinese or Thai, for example), there is not a special symbol to delimit words (such as the blank in most non-Asian writing systems). Current systems usually work around this problem by using a segmentation tool to pre-process the data. There are however two major disadvantages:

- These tools usually need a lot of linguistic knowledge, such as lexical dictionaries and hand-crafted segmentation rules. So using them somehow reduces the “purity” and universality of the statistical approach.
- These tools are not perfect. They tend to be very dependent on the domain of the text they are used with. Besides, they cannot take advantage of the fact that there exist a translation of the sentence in another language.

(Xu, Zens and Ney,2004) have overcome part of these objections by using multiple segmentations of a Chinese sentence and letting a SMT system choose the best one, as well as creating a segmentation lexicon dictionary by considering every Chinese character to be a word in itself and then creating a phrase alignment. However, it is probable that this technique would meet much more difficulties with Thai, for example (whose characters, unlike Chinese, bear no specific sense) or even Japanese (which use both ideograms and phonetic characters).

Besides, even for more “computer-friendly” languages, relying too much on typographic words may not be the best way to create an alignment. For example, the translation of a set phrase may contain no word that is a translation of the individual words of this set phrase. And one could consider languages such as German, which tend to merge words that are in relation in a single typographic word. For such languages, it could be a good thing to be able to create alignment at an even more basic level than the typographic words.

These thoughts are the main motivations for the development of the alignment algorithm we will expose in this paper. Its main advantage is that it can be applied whatever is the smallest unit of text we want to consider: typographic word or single character. And even when working at the character level, it can use larger sequence of characters to create correct alignments. The problem of the segmentation and of the alignment will be resolved simultaneously: a sentence and its translation will mutually induce a segmentation on one another. Another advantage of this algorithm is that it is purely statistical: it will not require any information other than the parallel corpus we want to align.

It should be noted here that the phrase-level joint-probability model presented in (Marcu and Wong) can pretend to have the same qualities. However, it was only applied to word-segmented texts by its authors. Making use of the EM training, it is also much more complex than our approach.

Before describing our algorithm, we will explain in detail a method for extracting the co-occurrence counts of any substring in a parallel corpus. Such co-occurrence counts are important to our method, but difficult to compute or store in the case of big corpora.

2 Co-Occurrence counting algorithm

2.1 Notation and definitions

In the subsequent parts of this paper, a substring will denote indifferently a sequence of characters or a sequence of words (or actually a sequence of any typographic unit we might want to consider). The terms "elements" will be used instead of "word" or "characters" to denote the fundamental typographic unit we chose for a given language.

In general, the number of co-occurrences of two substrings S_1 and S_2 in a parallel corpus is the number of times they have appeared on the opposite sides of a bi-sentence in this corpus. It will be noted $N(S_1,S_2)$. In the cases where S_1 and S_2 appears several times in a single bi-sentence (n_1 and n_2 times respectively), we might count 1, n_1*n_2 or $min(n_1,n_2)$ co-occurrences. We will also note $N(S_1)$ the number of occurrences of S_1 in the corpus.

2.2 The Storage Problem

Counting word co-occurrences over a parallel corpus and storing them in a data structure such as a Hash table is a trivial task. But storing the co-occurrences counts of every pair of substring presents much more technical difficulties. Basically, the problem is that the number of values to be stored is much greater when we consider substrings. For two sentences with N_1 and N_2 words respectively, there are N_1*N_2 words that co-occur; but the number of substrings that co-occur is roughly proportional to (N_1*N_2)^2. Of course, most substrings in a pair of sentences are not unique in the corpus, which reduces the number of values to be stored. Still, in most cases, it remains impractical. For example, the Japanese-English BTEC corpus has more than 11 million unique English (word-) substrings and more than 8 million unique Japanese (character-) substrings. So there are potentially 88,000 billion co-occurrence values to be stored. Again, most of these substrings do not co-occur in the corpus, so that non-zero co-occurrences values are only a fraction of this figure. However, a rough estimation we performed showed that there still would be close to a billion values to store.

With a bigger corpus such as the European Parliament Corpus (more than 600,000 sentences per languages) we have more than 698 millions unique English (word-) substrings and 875 millions unique French (word-) substrings. And things get much worse if we want to try to work with characters substrings.

To handle this problem, we decided not to try and store the co-occurrences count beforehand, but rather to compute them "on-the-fly", when they are needed. For that we will need a way to compute co-occurrences very efficiently. We will show how to do it with the data structure known as Suffix Array.

2.3 Suffix Arrays

Suffix Arrays are a data structure allowing for (among other things) the efficient computation of the number of occurrences of any substring within a text. They have been introduced by Mamber and Myers (1993) in a bioinformatics context. (Callison-Burch, Bannard and Scroeder, 2005) used them (in a way similar to us) to compute and store phrase translation probabilities over very large corpora.

Basically, a Suffix Array is a very simple data structure: it is the sorted list of all the suffixes of a text. A suffix is a substring going from one starting position in the text to its end. So a text of T elements has T suffixes.

An important point to understand is that we won't have to store the actual suffixes in memory. We can describe any suffix by its starting position in the text. Hence, every suffix occupies a constant space in memory. Actually, a common implementation is to represent a suffix by a memory pointer on the full text. So, on a machine with 32-bit pointers, the Suffix Array of a text of T elements occupy $4*T$ bytes. The time complexity of the Suffix Array construction is $O(T*log(T))$ if we build the array of the suffixes and then sort it.

We will now describe the property of the Suffix Array that interest us. Let S be a substring. Let pf be the position (in the Suffix Array) of the first suffix beginning with substring S and pl be the position of the last such suffix. Then every suffix in the Array between positions pf and pl corresponds to an occurrence of S. And every occurrence of S in the text corresponds to a suffix between pf and pl.

pf and pl can be retrieved in $O(|S|*log\ T)$ with a dichotomy search. Beside, $N(S)=pl-pf+1$; so we can compute $N(S)$ in $O(|S|*log\ T)$. We will now see how to compute $N(S_1,S_2)$ for two substrings S_1 and S_2 in a parallel corpus.

2.4 Computing Co-Occurrences using Suffix Array

A Suffix Array can be created not only from one text, but also from a sequence of texts. In the present case, we will consider the sequence of sentences formed by one side of a parallel corpus. The Suffix Array is then the sorted list of all the suffixes of all the sentences in the sequence. Suffixes may be represented as a pair of integer (*index of the sentence*, *position in the sentence*) or again as a pointer (an example using integer pairs is shown on Figure 1).

We can implement the Suffix Array so that, from a suffix, we can determine the index of the sentence to which it belongs (the computational cost of this is marginal in practical cases and will be neglected). We can now compute pf and pl for a substring S such as previously, and retrieve the sentence indexes corresponding to every suffix between positions pf and pl in the Suffix Array, This allow us to create an "*occurrence vector*": a mapping between sentence indexes and the number of occurrences of S in those sentences. This operation takes $O(pl-pf)$, that is $O(N(S))$. (Figure 1. shows an occurrence vector for the substring "red car")

We can now efficiently compute the co-occurrence counts of two substrings S_1 and S_2 in a parallel corpus.

We compute beforehand the two Suffix Arrays for the 2 sides of the parallel corpus. We can then compute two occurrence vectors V_1 and V_2 for S_1 and S_2 in $O(N(S_1)+|S_1|*log(T_1))$ and $O(N(S_2)+|S_2|*log(T_2))$ respectively.

A small monolingual corpus	
index	sentence
1	The red car is here
2	I saw a blue car
3	I saw a red car

Occurrence Vector of "**red car**"	
index	nbOcc
1	1
2	0
3	1

Suffix Array		
Array index	Suffix Position	Suffix
0	2,3	a blue car
1	3,4	a red car
2	2,4	blue car
3	2,6	car
4	3,5	car
5	1,3	car is here
6	1,5	here
7	2,1	I saw a blue car
8	1,1	I saw a red car
9	1,4	is here
10	**1,2**	**red car is here**
11	**3,5**	**red car**
12	2,2	saw a blue car
13	3,3	saw a red car
14	1,1	The red car is here

Figure 1. A small corpus, the corresponding suffix array, and an occurrence vector

With a good implementation, we can use these two vectors to obtain $N(S_1,S_2)$ in $O(min(size(V_1),size(V_2)))$, that is $O(min(N(S_1),N(S_2)))$.

Hence we can compute $NbCoOcc(S_1,S_2)$ for any substring pair (S_1,S_2) in $O(N(S_2)+|S_2|*log(T_2)+N(S_1)+|S_1|*log(N_1)))$. This is much better than a naive approach that takes $O(T_1*T_2)$ by going through the whole corpus. Besides, some simple optimizations will substantially improve the average performances.

2.5 Some Important Optimizations

There are two ways to improve performances when using the previous method for co-occurrences computing.

Firstly, we won't compute co-occurrences for any substrings at random. Typically, in the algorithm described in the following part, we compute $N(S_1,S_2)$ for every substring pairs in a given bi-sentence. So we will compute the occurrence vector of a substring only once per sentence.

Secondly, the time taken to retrieve the co-occurrence count of two substrings S_1 and S_2 is more or less proportional to their frequency in the corpus. This is a problem for the average performance: the most frequent substrings will be the one that take longer to compute. This suggests that by caching the occurrence vectors of the most frequent substrings (as well as their co-occurrence counts), we might expect a good improvement in performance. (We will see in the next sub-section that caching the 200 most frequent substrings is sufficient to multiply the average speed by a factor of 50)

2.6 Practical Evaluation of the Performances

We will now test the computational practicality of our method. For this evaluation, we will consider the English-Japanese BTEC corpus (170,000 bi-sentences, 12MB), and the English-French Europarl corpus (688,000 bi-sentences, 180 MB). We also want to apply our algorithm to western languages at the character level. However, working at a character level multiply the size of the suffix array by about 5, and increase the size of the cached vectors as well. So, because of memory limitations, we extracted a smaller corpus from the Europarl one (100,000 bi-sentences, 20MB) for experimenting on characters substrings.

The base elements we will choose for our substrings will be: word/characters for the BTEC, word/word for the bigger EuroParl, and word/characters for the smaller EuroParl. We computed the co-occurrence counts of every substrings pair in a bi-sentence for the 100 first bi-sentences of every corpus, on a 2.5GHz x86 computer. We give the average figures for different corpora and caching strategies.

Corpus	Cache (cached substrg)	Allocated Memory (MB)	CoOcc computed (per sec.)	bisentences processed (per sec.)
BTEC	0	22	7k	1.2
BTEC	200	120	490k	85
EuroParl	0	270	3k	0.4
EuroParl	400	700	18k	1.2
Small EuroParl	0	100	4k	0.04
Small EuroParl	400	300	30k	0.3

These results are good enough and show that the algorithm we are going to introduce is not computationally impracticable. The cache allows an interesting trade-off between the performances and the used memory. We note that the proportional speedup depends on the corpus used. We did not investigate this point, but the different sizes of corpora (inducing different average *occurrence vectors* sizes), and the differences in the frequency distribution of words and characters are probably the main factors.

3 Sub-sentential alignment

3.1 The General Principle

Given two substrings S_1 and S_2, we can use their occurrence and co-occurrence counts to compute a correlation coefficient (such as the chi-square statistic, the point-wise mutual information or the Dice coefficient).

The basic principle of our sub-sentential alignment algorithm will simply be to compute a correlation coefficient between every substring in a bi-sentence, and align the substrings with the highest correlation. This idea needs, however, to be refined.

First, we have to take care of the indirect association problem. The problem, which was described in (Melamed, 1997) in a word-to-word alignment context, is as follows: if e_1 is the translation of f_1 and f_2 has a strong monolingual association with f_1, e_1 and f_2 will also have a strong correlation. Melamed assumed that indirect associations are weaker than direct ones, and provided a Competitive Linking Algorithm that does not allow for a word already aligned to be linked to another one. We will make the same assumption and apply the same solution. So our algorithm will align the substring pairs with the highest correlation first, and will forbid the subsequent alignment of substrings having a part in common with an already aligned substring. A side-effect of this procedure is that we will be constrained to produce a single segmentation on both sentences and a single alignment between the components of this segmentation. According to the application, this might be what we are looking for or not. But it must be noted that, most of the time, alignments with various granularities are possible, and we will only be able to find one of them. We will discuss the issue of the granularity of the alignment in part **3.3**.

Besides, our approach implicitly considers that the translation of a substring is a substring (there are no discontinuities). This is of course not the case in general (for example, the English word "*not*" is usually translated in French by "*ne…pas*"). However, there is most of the time a granularity of alignment at which there is no discontinuity in the alignment components.

Also, it is frequent that a word or a sequence of words in a sentence has no equivalent in the opposite sentence. That is why it will not be mandatory for our algorithm to align every element of the sentences at all cost. If, at any point, the substrings that are yet to be linked have correlation coefficients below a certain threshold, the algorithm will not go further.

So, the algorithm can be described as follow:

1- Compute a correlation coefficient for all the substrings pairs in e and f and mark all the elements in e and f as free.

2- Among the substrings which contain only free element, find the pair with the highest correlation. If this correlation is not above a certain threshold, end the algorithm. Else, output a link between the substrings of the pair.

3- Mark all the elements belonging to the linked pair as non-free.

4- Go back to 2

It should be noted that correlation coefficients are only meaningful data is sufficiently available; but many substrings will appear only a couple of times in the corpus. That is why, in our experiments we have set to zero the correlation coefficient of substring pairs that co-occur less than 5 times (this might be a bit conservative, but the BTEC corpus we used being very redundant, it was not too much of a restriction).

3.2 Giving a preference to bigger alignments.

A problem that arose in applying the previous algorithm is a tendency to link incomplete substrings. Typically, this happen when a substring S_1 can be translated by two substrings S_2 and S_2', S_2 and S_2' having themselves a common substring. S_1 will then be linked to the common part of S_2 and S_2'. For example, the English word "museum" has two Japanese equivalents: 博物館 and 美術館. In the BTEC corpus, the common part (館) will have a stronger association with "museum", and so will be linked instead of the correct substring (博物館 or 美術館).

To prevent this problem, we have tried to modify the correlation coefficients so that they slightly penalize shorter alignment. Precisely, for a substring pair (S_1,S_2), we define its area as *"length of S_1"*"length of S_2"*. We then multiply the Dice coefficient by $area(S_1,S_2)$ and the chi-square coefficient by $log(area(S_1,S_2)+1)$. These formulas are very empiric, but they created a considerable improvement in our experimental results.

Linking the bigger parts of the sentences first has another interesting effect: bigger substrings present less ambiguity, and so linking them first may prevent further ambiguities to arise. For example, with the bi-sentence *"the cat on the wall"/"le chat sur le mur"*. Each *"the"* in the English sentence will have the same correlation with each *"le"* in the French sentence, and so the algorithm cannot determine which *"the"* correspond to which *"le"*. But if, for example *"the cat"* has been previously linked to *"le chat*", there is no more ambiguity.

We mentioned previously the issue of the granularity of alignments. These "alignment size penalties" could also be used to tune the granularity of the alignment produced.

3.3 Experiments and Evaluations

Although we made some tests to confirm that computation time did not prevent our algorithm to work with bigger corpus such as the EuroParl corpus, we have until now limited deeper experiments to the Japanese-English BTEC Corpus.

That is why we will only present results for this corpus. For comparison, we re-implemented the ISA (Integrated Segmentation Alignment) algorithm described in (Zhang, Vogel and Waibel, 2003). This algorithm is interesting because it is somehow similar to our own approach, in that it can be seen as a generalization of Melamed's Competitive Linking Algorithm. It is also fairly easy to implement. A comparison with the joint probability model of Marcu and Wong (which can also work at the phrase/substring level) would have also been very interesting, but the difficulty of implementing and adapting the algorithm made us delay the experiment.

After trying different settings, we chose to use chi-square statistic as the correlation coefficient for the ISA algorithm, and the dice coefficient for our own algorithm. ISA settings as well as the "alignment size penalties" of our algorithm were also tuned to give the best results possible with our test set. For our algorithm, we considered word-substrings for English and characters substrings for Japanese. For the ISA algorithm, we pre-segmented the Japanese corpus, but also tried to apply it directly to Japanese by considering characters as words.

Estimating the quality of an alignment is not an easy thing. We tried to compute a precision and a recall score in the following manner. Precision was such that:

$$Precision = \frac{Nb\ of\ correct\ links}{Nb\ of\ outputted\ links}$$

Correct link are counted by manual inspection of the results. Appreciating what is a correct link is subjective; especially here, where we consider many-words-to-many-characters links. Overall, the evaluation was pretty indulgent, but tried to be consistent, so that the comparison would not be biased.

Computing recall is more difficult: for a given bi-sentence, multiple alignments with different granularities are possible. As we are only trying to output one of these alignments, we cannot define easily a "gold standard". What we did was to count a missed link for every element that was not linked correctly and could have been. We then compute a recall measure such that:

$$Recall = \frac{Nb\ of\ correct\ links}{Nb\ of\ correct\ links + Nb\ of\ missed\ links}.$$

These measures are not perfect and induce some biases in the evaluation (they tend to favor algorithms aligning bigger part of the sentence, for example), but we think they still give a good summary of the results we have obtained so far.

As can be seen in the following table, our algorithm performed quite well. We are far from the results obtained with a pre-segmentation, but considering the simplicity of this algorithm, we think these results are encouraging and justify our initial ideas. There is still a lot of room for improvement: introducing a n-gram language model, using multiple iterations to re-estimate the correlation of the substrings...

That is why we are pretty confident that we can hope to compete in the end with algorithms using pre-segmentation.

	Precision	Recall
Our algorithm (w/o segmentation)	78%	70%
ISA (w/o segmentation)	55%	55%
ISA + segmentation	98%	95%

Also, although we did not make any thorough evaluation, we also applied the algorithm to a subset of the Europarl corpus (cf. **2.6**), where characters where considered the base unit for French. The alignments were mostly satisfying (seemingly better than with the BTEC). But hardly any sub-word alignments were produced. Some variations on the ideas of the algorithm, however, allowed us to get interesting (if infrequent) results. For example, in the pair (‘I would like’/ ‘Je voudrais’), ‘would’ was aligned with ‘rais’ and ‘voud’ with ‘like’.

4 Conclusion and future work

In this paper we presented both a method for accessing the co-occurrences count for any substring pair in a parallel corpus and an algorithm taking advantage of this method to create subsentential alignments in such a corpus.

We showed our co-occurrence counting method performs well with corpus commonly used in Statistical Machine Translation research, and so we think it can be a useful tool for the statistical processing of parallel corpora.

Our phrase level alignment algorithm gave encouraging results, especially considering there are many possibilities for further improvement.

In the future, we will try to improve the algorithm as well as perform more extensive evaluations on different language pairs.

References

Ralph Brown. 1997. *Automated Dictionary Extraction for Knowledge-Free Example Based Translation*, Proceedings of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation, pp. 111-118, Santa-Fe, July 1997.

Chris Callison-Burch, Colin Bannard and Josh Scroeder. 2005. *Scaling Phrase-Based Statistical Machine Translation to Larger Corpora and Longer Phrases*, Proceedings of 43rd Conference of the Association for Computational Linguistics (ACL 05), Ann Arbor, USA, 2005.

Philipp Koehn. 2003. *Europarl: A Multilingual Corpus for Evaluation of Machine Translation*, Draft,Unpublished.

Manber and Myers. 1993. *Suffix Array: A New Method for On-Line String Searches*, SIAM Journal on Computing, 22(5):935-948.

Daniel Marcu, William Wong. 2002. *A Phrase-Based, Joint Probability Model for Statistical Machine Translation*, Proceedings of the Conference on Empirical Methods in Natural Language Processing , Philadelphia, USA, July 2002.

Dan Melamed. 1997. *A Word-to-Word Model of Translational Equivalence*, Proceedings of 35th Conference of the Association for Computational Linguistics (ACL 97), Madrid, Spain, 1997.

Franz Joseph Och, Christophe Tillmann, Hermann Ney. 1999. *Improved Alignment Models for Statistical Machine Translation*. Proceedings of the joint conference of Empirical Methods in Natural Language Processing and Very Large Corpora, pp 20-28, University Of Maryland,

Jia Xu, Richard Zens., Hermann Ney. 2004. *Do We Need Chinese Word Segmentation for Statistical Machine Translation?*, Proceedings of the 3rd SIGHAN Workshop on Chinese Language Learning, Barcelona, Spain, pp. 122-128 , July 2004

Ying Zhang, Stephan Vogel and Alex Waibel. 2003. *Integrated Phrase Segmentation and Alignment algorithm for Statistical Machine Translation*, Proceedings of International Conference on Natural Language Processing and Knowledge Engineering, Beijing,China., October 2003