Enhancing the Bilingual Concordancer TransSearch with Word-level Alignment

Julien Bourdaillet, Stéphane Huet, Fabrizio Gotti,
Guy Lapalme, and Philippe Langlais

Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
C.P. 6128, succursale Centre-ville
H3C 3J7, Montréal, Québec, Canada
http://rali.iro.umontreal.ca

Abstract. Despite the impressive amount of recent studies devoted to improving the state of the art of Machine Translation (MT), Computer Assisted Translation (CAT) tools remain the preferred solution of human translators when publication quality is of concern. In this paper, we present our perspectives on improving the commercial bilingual concordancer `TransSearch`, a Web-based service whose core technology mainly relies on sentence-level alignment. We report on experiments which show that it can greatly benefit from statistical word-level alignment.

1 Introduction

Although the last decade has witnessed an impressive amount of effort devoted to improving the current state of Machine Translation (MT), professional translators still prefer Computer Assisted Translation (CAT) tools, particularly *translation memory* (TM) systems. A TM is composed of a *bitext*, a set of pairs of units that are in translation relation, plus a search engine. Given a new text to translate, a TM system automatically segments the text into units that are systematically searched for in the memory. If a match is found, the associated target material is retrieved and output with possible modifications, in order to account for small divergences between the unit to be translated and the one retrieved. Thus, such systems avoid the need to re-translate previously translated units. Commercial solutions such as `SDL Trados`[1], `Deja Vu`[2], `LogiTerm`[3] or `MultiTrans`[4] are available; they mainly operate at the level of sentences, which narrows down their usefulness to repetitive translation tasks.

Whereas a TM system is a translation device, a *bilingual concordancer* (BC) is conceptually simpler, since its main purpose is to retrieve from a bitext, the pairs of units that contain a *query* (typically a phrase) that a user manually

[1] http://www.trados.com
[2] http://www.atril.com
[3] http://www.terminotix.com
[4] http://www.multicorpora.ca

submits. It is then left to the user to locate the relevant material in the retrieved target units. As simple as it may appear, a bilingual concordancer is nevertheless a very popular CAT tool. In [1], the authors report that `TransSearch`,[5] the commercial concordancer we focus on in this study, received an average of 177 000 queries a month over a one-year period (2006–2007).

This study aims at improving the current `TransSearch` system by providing it with robust word-level alignment technology. It was conducted within the `TS3` project, a partnership between the RALI and the Ottawa-based company Terminotix.[6] One important objective of the project is to automatically identify (highlight) in the retrieved material the different translations of a user query, as discussed initially in [2]. The authors of that paper also suggested that grouping variants of the same "prototypical" translation would enhance the usability of a bilingual concordancer. These are precisely the two problems we are addressing in this study.

The remainder of this paper is organized as follows. We first describe in Section 2 the translation spotting techniques we implemented and compared. Since translation spotting is a notoriously difficult problem, we discuss in Section 3 two novel issues that we think are essential to the success of a concordancer such as `TransSearch`: the identification of erroneous alignments (Section 3.1) and the grouping of translation variants (Section 3.2). We report on experiments in Section 4 and conclude our discussion and propose further research avenues in Section 5.

2 Transpotting

Translation spotting, or *transpotting*, is the task of identifying the word-tokens in a target-language (TL) translation that correspond to the word-tokens of a query in a source language (SL) [3]. It is therefore an essential part of the `TS3` project. We call *transpot* the target word-tokens automatically associated with a query in a given pair of units (sentences). The following example[7] illustrates the output of one of the transpotting algorithms we implemented. Both `conformes à` and `fidèles à` are French transpots of the English query `in keeping with`.

S_1 = `These are important measures in keeping with our international obligations.`
T_1 = `Il s'agit d'importantes mesures conformes à nos obligations internationales.`

S_2 = `In keeping with their tradition, liberals did exactly the opposite.`
T_2 = `Fidèles à leur tradition, les libéraux ont fait exactement l'inverse.`

[5] `www.tsrali.com`
[6] `www.terminotix.com`
[7] The data used in this study is described in Section 4.1.

As mentioned in [4], translation spotting can be seen as a by-product of word-level alignment. Since the seminal work of [5], statistical word-based models are still the core technology of today's Statistical MT. This is therefore the alignment technique we consider in this study.

Formally, given a SL sentence $S = s_1...s_n$ and a TL sentence $T = t_1...t_m$ in translation relation, an IBM-style alignment $a = a_1...a_m$ connects each target token to a source one ($a_j \in \{1, ..., n\}$) or to the so-called NULL token which accounts for untranslated target tokens, and which is arbitrarily set to the source position 0 ($a_j = 0$). This defines a word-level alignment space between S and T whose size is in $O(m^{n+1})$.

Several word-alignment models are introduced and discussed in [5]. They differ by the expression of the joint probability of a target sentence and its alignment, given the source sentence. We focus here on the simplest form, which corresponds to IBM models 1 & 2:

$$p(t_1^m, a_1^m) = \prod_{j=1}^{m} \sum_{i \in [0,n]} p(t_j|s_i) \times p(i|j, m, n)$$

where the first term inside the summation is the so-called transfer distribution and the second one is the alignment distribution.

A transpotting algorithm comprises two stages: an alignment stage and a decision stage. We describe in the following sections the algorithms we implemented and tested, with the convention that $s_{i_1}^{i_2}$ stands for the source query (we only considered contiguous queries in this study, since they are by far the most frequent according to our user logfile).

2.1 Viterbi Transpotting

One straightforward strategy is to compute the so-called Viterbi alignment ($\hat{a}$). By applying Bayes' rule and removing terms that do not affect the maximization, we have:

$$\hat{a} = \underset{a_1^m}{\operatorname{argmax}}\, p(a_1^m|t_1^m, s_1^n) = \underset{a_1^m}{\operatorname{argmax}}\, \frac{p(s_1^n) \times p(t_1^m, a_1^m|s_1^n)}{p(s_1^n) \times p(t_1^m|s_1^n)} = \underset{a_1^m}{\operatorname{argmax}}\, p(t_1^m, a_1^m|s_1^n)$$

In the case of IBM models 1 & 2, this alignment is computed efficiently in $O(n \times m)$. Our first transpotting implementation, called VITERBI-SPOTTER, simply gathers together the words aligned to each token of the query. Note that with this strategy, nothing forces the transpot of the query to be a contiguous phrase. In our first example above, the transpot of the query produced by this formula for `in keeping with` is the French word `conformes`.

2.2 Contiguous Transpotting

This method, which was introduced by M. Simard in [4], forces the transpot to be a contiguous sequence. This is accomplished by computing for each pair

$\langle j_1, j_2 \rangle \in [1, m]^2$, two Viterbi alignments: one between the phrase $t_{j_1}^{j_2}$ and the query $s_{i_1}^{i_2}$, and one between the remaining material in those sentences, $\bar{s}_{i_1}^{i_2} \equiv s_1^{i_1-1} s_{i_2+1}^{n}$ and $\bar{t}_{j_1}^{j_2} \equiv t_1^{j_1-1} t_{j_2+1}^{m}$. The complexity of this algorithm is $O(nm^3)$:

$$\hat{t}_{\hat{j}_1}^{\hat{j}_2} = \underset{(j_1,j_2)}{\operatorname{argmax}} \left\{ \max_{a_{j_1}^{j_2}} p(a_{j_1}^{j_2} | s_{i_1}^{i_2}, t_{j_1}^{j_2}) \times \max_{\bar{a}_{j_1}^{j_2}} p(\bar{a}_{j_1}^{j_2} | \bar{s}_{i_1}^{i_2}, \bar{t}_{j_1}^{j_2}) \right\}$$

This method, called ContigVit-spotter hereafter, returns the transpot `conformes à` for the query `in keeping with` in the first example above.

2.3 Maximum Contiguous Subsequence transpotting

In this transpotting strategy, called MCSS-spotter, each token t_j is associated with a score computed such as:

$$\text{score}(t_j) = \text{score}_0(t_j) - \tilde{t}$$

where $\text{score}_0(t_j) = \sum_{i=i_1}^{i_2} p(t_j|s_i) p(i|j,m,n)$ and $\tilde{t} = \frac{1}{m} \sum_{j=1}^{m} \text{score}_0(t_j)$.

The score corresponds to the word alignment score of IBM model 2 minus the average computed for each token t_j. Because of $\tilde{t}$, the score associated with a token t_j is either positive or negative.

The Maximum Contiguous Subsequence Sum (MCSS) algorithm is then applied. Given this sequence of scores, it finds the contiguous subsequence whose sum of scores is maximum over all subsequences. When processing the sequence, the trick is that if a contiguous subsequence with a negative sum is encountered, it cannot be a MCSS; therefore, either the MCSS occurred before this negative sum subsequence or it will occur after. Our implementation runs in $O(m)$. Finally, the MCSS corresponds to the transpot of the given query $s_{i_1}^{i_2}$. In the first example above, the unsatisfying transpot `à` is returned by this method.

2.4 Baseline

To challenge the transpotting algorithms we implemented, we also considered a strategy which does not embed any statistical alignment. It consists in projecting the positions of the query $s_{i_1}^{i_2}$ by means of the length ratio between the two sentences. The transpot is determined by $t_{j_1} \dots t_{j_2}$ where $j_1 = \lfloor \frac{m}{n} i_1 \rfloor$ and $j_2 = \lceil \frac{m}{n} i_2 \rceil$. In our example, this method, called Baseline-spotter hereafter, returns the transpot `importantes mesures conformes à`.

3 Post-Processing

Frequent queries in the translation memory receive numerous translations by the previously described transpotting process. Figure 1 illustrates the many transpots returned by ContigVit-spotter for the query `in keeping with`. As can

be observed, some transpots (those marked by a star) are clearly wrong (*e.g.* `à`), while many others (in italics) are only partially correct (*e.g.* `conformément`). Also, it appears that many transpots are indeed very similar (*e.g.* `conforme à` and `conformes à`).

Since in `TS3` we want to offer the user a list of retrieved translations for a query, strategies must be devised for overcoming alignment errors and delivering the most salient information to the user. We investigated two avenues in this study: detecting erroneous transpots (Section 3.1) and merging variants of the same prototypical translation (Section 3.2).

`conforme à` (45)	`conforme aux` (18)	`conformes à` (11)
`conformément à` (29)	*`conforme`* (13)	*`conformément`* (9)
`à`* (21)	`conformément aux` (13)	`, conformément à` (3)
`dans`* (20)	`conforme au` (12)	`correspondant à` (3)

Fig. 1. Subset of the transpots retrieved by ContigVit-spotter for the query `in keeping with` with their frequency shown in parentheses.

3.1 Refining Transpotting

As detailed in Section 4.1 below, we analyzed 531 queries and their transpots, as computed by ContigVit-spotter, and manually annotated the erroneous transpots. This corpus served to train a classifier designed to distinguish good transpots from bad ones. To this end, we applied the *voted-perceptron* algorithm described in [6]. Online voted-perceptrons have been reported to work well in a number of NLP tasks [7, 8]. In a nutshell, a weighted pool of perceptrons is incrementally acquired during a batch training procedure, where each perceptron is characterized by a real-valued vector (one component per feature on which we train the classifier) and its associated weight, computed as the number of successive training examples it could correctly classify before it fails. When the current perceptron misclassifies a training example, a new one is added to the pool, the coefficients of which are initialized from the current perceptron according to a simple delta-rule and kept fixed over the training procedure.

We computed three groups of features for each example of the annotated corpus, that is, each query/transpot (q, t) pair. The first group is made up of features related to the size (counted in words) of q and t, with the intuition that they should be related. The second group gathers various alignment scores computed with word-alignment models (min and max likelihood values, etc.). The last group gathers clues that are more linguistically flavored, among them the ratio of grammatical words in q and t, or the number of prepositions and articles. In total, each example is represented by at most 40 numerical features.

3.2 Merging Variants

Once erroneous transpots have been filtered out, there usually remain many translation variants for a given query. Some of them are very similar and are therefore redundant for the user. For instance, returning the inflected forms of nouns or verbs is often useless and may prevent more dissimilar and potentially more interesting variants from being shown to the user when the number of displayed translations for a query is limited. This phenomenon is more acute for the French language with its numerous verb conjugation forms. Another problem that often shows up is that many transpots differ only by punctuation marks or by a few grammatical words, *e.g.* `conformément aux` and `, conformément à` in Figure 1.

Merging variants according to their closeness raises several difficulties. First, the various transpots must be compared, which represents a costly process. Second, we need to identify clusters of similar variants. Lastly, a prototype of the selected clusters must be selected and output to the user. We now describe our solution to these problems.

Comparing the transpots pairwise is an instance of multiple sequence alignment, a well studied problem in bioinformatics [9]. We adopt the approach of progressive alignment construction. This method first computes the distance between each pair of transpots to align and progressively builds a tree that aims at guiding the alignment of all pairs. At each step, the most similar pair is merged and added to the tree, until no transpot remains unaligned. In order to build this tree, we use a bottom up clustering method, called *neighbor-joining* [10].

The main interest of this approach is its computational efficiency, since pairwise aligning the transpots is carried out in polynomial time, which allows us to use it even when a large set of transpots is returned. This property is obtained thanks to the greedy nature of the algorithm. Indeed, it is based on a metrics that can be straightforwardly computed between a new node —associated with a joined pair of sequences— and the other sequences from the metrics previously computed for the sequences just joined. Although this clustering method is greedy and may not build the optimal tree, it has been extensively tested and usually finds a tree that is quite close to the optimal one.

The neighbor-joining algorithm requires computing a distance matrix between each pair of transpots to align. A word-level specific edit-distance was empirically developed to meet the constraints of our application. Different substitution, deletion or insertion costs are introduced according to the grammatical classes or possible inflections of the words; it is therefore language dependent. We used an in-house lexicon which lists for both French and English the lemmas of each word-form and its possible part-of-speech. A minimal substitution cost was empirically engineered between two inflected forms of the same lemma. An increasing edition cost was set empirically to account respectively for punctuation marks, articles, grammatical words (prepositions, conjunctions and pronouns), auxiliary verbs and finally all the remaining words (verbs, nouns, adjectives and adverbs).

Thus, we obtain a tree whose leaves are transpots. The closest leaves in the tree correspond to the closest variants, according to our edit-distance calculation. Therefore, clusters of similar variants can be formed by traversing the tree in a post-order manner. The transpots which are associated with two neighboring leaves and which differ only by grammatical words or by inflectional variants are considered as sufficiently similar to be merged into a cluster. This process is repeated until all the leaves have been compared with their nearest neighbor and no more similar variants are found. For each pair of merged leaves, a pattern is built from the alignment of the two transpots and regular expressions are used to represent the grouped variants.

Figure 2 illustrates this process with an extract from the output obtained for the examples in Figure 1. `conforme à` and `conformes à` are neighboring transpots in the tree which are grouped into the pattern `[conforme] à` and added to the tree. Similarly, the prototype `conforme [au]` is built from the two neighboring transpots `conforme au` and `conforme aux`. Once these merges are done, the two new prototypes become close in the tree; their comparison in turn leads to the decision to group them and to create the pattern `[conforme] [à|au]`.

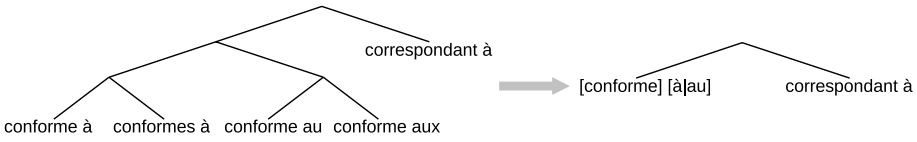

Fig. 2. Merging of close transpots by the process described in the text.

4 Experiments

4.1 Corpora

Translation memory The largest collections in `TransSearch` come from the Canadian Hansards, that is, parallel texts in English and French drawn from official records of the proceedings of the Canadian Parliament. This material is aligned at the sentence-level by an in-house aligner. For our experiments, we indexed with Lucene[8] a translation memory comprising 3.3 million pairs of French-English sentences. This was the maximum amount of material we could train a statistical word-alignment model on, running the `giza++` [11] toolkit on a computer equipped with 16 gigabytes of memory.

Automatic reference corpus In order to evaluate the quality of our approach, we developed a reference corpus called REF. In [4], the author manually identified

[8] `http://lucene.apache.org`

the transpots in 4 100 pairs of sentences produced by 41 queries, a slow and difficult process. Indeed, the time spent analyzing one query with 30 pairs of sentences to annotate was in the order of 5-10 minutes.

We devised a way to get a much larger reference corpus without manual annotation. The 5 000 most frequent queries submitted by users to the system were extracted from the logs of `TransSearch`. Besides, we used an in-house bilingual phrase dictionary collected for the needs of various projects, which includes 59 057 English phrases with an average of 1.4 French translations each. Among the indexed pairs of sentences, only those that contain the phrases of the dictionary and their translation are kept.

According to this method, 4 526 of the 5 000 most frequent queries submitted by users to the system actually occured in our translation memory; of these 2 130 had a translation either in the bilingual phrase dictionary (713) or in a classical bilingual word dictionary (the remaining). From the memory, we retrieved a maximum of 5 000 pairs of sentences for each of those 2 130 queries, leading to a set of 1 102 357 pairs of sentences, with an average of 517 pairs of sentences per query. Altogether, these examples contain 7 472 unique pairs of query/transpot; each query received an average of 3.5 different transpots, and a maximum of 37.

Human reference In order to train the classifier described in Section 3.1, four human annotators (a subset of the authors) were asked to identify bad transpots among those proposed by the best of our transpotting algorithm. We decided to annotate the query/transpot pairs without their contexts. This allows a relatively fast annotation process, in the order of 40 seconds per query, but leaves some cases difficult to annotate. To go back to the query `in keeping with`, though some translations like `conforme à` are straightforward, other such as `suivant`, `dans le sens de` or even `tenir compte de` can be the correct transpots of this query according to its context of use.

We ended up with a set of 531 queries that have an average of 22.9 transpots each, for a total of 12 144 annotated examples. We computed the inter-annotator agreement and observed a 0.76 kappa score [12], which indicates a high degree of agreement among annotators.

4.2 Translation Spotting

To compare our transpotting algorithms, we conducted two series of evaluation: one at the sentence level, and one at the query level. In the former case, the ability of each algorithm to identify the reference translation $\hat{t}$ for a query q was measured according to precision and recall ratios, computed as follows:

$$\text{precision} = |t \cap \hat{t}|/|t| \qquad \text{recall} = |t \cap \hat{t}|/|\hat{t}| \qquad \text{F-measure} = 2\frac{|t \cap \hat{t}|}{|t|+|\hat{t}|}$$

where t is the transpot identified by the algorithm, and the intersection operation is to be understood as the portion of words shared by t and $\hat{t}$. A point of detail is in order here: since several pairs of sentences can contain a given query/reference transpot pair $(q, \hat{t})$, we averaged the aforementioned ratios measured per unique

pairs $(q, \hat{t})$.[9] This avoids biasing our evaluation metrics toward frequent pairs in REF. Those average ratios are then averaged over the set of different pairs $(q, \hat{t})$ in REF.

Table 1 shows the results obtained by the different methods. We observe that MCSS-SPOTTER and CONTIGVIT-SPOTTER obtain the best results. MCSS-SPOTTER has a higher recall than CONTIGVIT-SPOTTER, meaning that its transpots t match more of the words of the references $\hat{t}$, but it has also a lower precision, meaning that its transpots are longer. A caricature of this strategy would be to propose the whole sentence as a transpot. This is very undesirable at the sentence level where transpotting must highlight with precision a transpot in a sentence. Finally, the results of CONTIGVIT-SPOTTER are more balanced: the behavior of this transpotter is more in keeping with what is expected. We comment on the last line of Table 1 in Section 4.3.

Table 1. Results of the different transpotting algorithms presented in Section 2 measured on the REF corpus.

method	precision	recall	F-measure
BASELINE-SPOTTER	0.127	0.222	0.149
VITERBI-SPOTTER	0.139	0.349	0.190
MCSS-SPOTTER	0.198	**0.744**	0.295
CONTIGVIT-SPOTTER	**0.303**	0.597	**0.376**
CONTIGVIT-SPOTTER + best voted-perceptron	**0.372**	**0.757**	**0.470**

While at the sentence-level evaluation, each pair of sentences containing a query and a reference translation counts, at the query-level, we directly evaluate the set of different transpots found for each query. On average, the CONTIGVIT-SPOTTER transpotting algorithm identifies 40 different transpots per query, and at least one reference translation was proposed for 91% of the queries.

4.3 Filtering Spurious Transpots

As described in Section 3.1, we trained various classifiers to identify spurious transpots. For this, 90 % of the human reference presented in Section 4.1 was used as a training corpus and 10 % as a test corpus. The examples (query/transpot pairs) are represented by three kinds of feature sets. All the classifiers, plus a few challenging baselines are evaluated according to the ratio of Correctly Classified Instances (CCI). Since in our application, we are interested in filtering out bad transpots, we also report precision, recall and F-measure rates related to this class. These figures are shown in Table 2.

To begin with, the simplest baseline we built (line 1) classifies all instances as good. This results in a CCI ratio of 0.62. A more sensible baseline that we

[9] Without this normalization, results would be increased by a range of 0.2 to 0.4 points.

engineered after we investigated the usefulness of different feature sets consists in classifying as bad those transpots whose ratio of grammatical words is above 0.75. It receives a CCI ratio of 0.81.

Among all voted-perceptron configurations we tested, with the exception of the one with all feature sets, the one making use of word alignment scores based on an IBM model 1 obtains the best CCI ratio of 0.82. Although this is barely better than the performance of the best baseline, the voted-perceptron shows a much higher gain in F-measure for bad transpots: 0.77 compared to 0.69, which is the task we are trying to optimize. Finally, the voted-perceptron trained using all feature sets (last line) obtains a CCI ratio of 0.84 and a F-measure for bad transpots of 0.78, which clearly surpasses the baseline. It should be noticed that while the best baseline has a better precision than the best voted-perceptron, precision and recall are more balanced for the latter. Because it is not clear whether precision or recall should be favored for the task of bad transpot filtering, we optimized the F-measure.

Table 2. Performance of different algorithms for identifying bad transpots using the test subset of the human reference.

			Bad		
		CCI	precision	recall	F-measure
Baselines:	all good	0.62	0.00	0.00	0.00
	grammatical ratio > 0.75	**0.81**	0.91	0.56	**0.69**
Features:	size	0.72	0.69	0.46	0.55
	IBM	**0.82**	0.75	0.80	**0.77**
	grammatical-ratio	0.80	0.90	0.55	0.68
	all	**0.84**	0.80	0.77	**0.78**

The last line of Table 1 presents the results of the best transpotter ContigVitspotter after filtering bad transpots with the best voted-perceptron. Significant gains can be observed: the F-measure increases from 0.376 to 0.470. It outperforms the MCSS-spotter recall score and has a higher precision of nearly 0.17. This demonstrates the interest of filtering bad transpots.

4.4 Merging Variants

The second post-processing stage, which merges variants, was evaluated on the pairs of sentences collected from the same 5 000 queries as those of the ref corpus. Contrary to the evaluation of transpot filtering, which requires a dictionary to build the reference, the retrieved pairs of sentences were not discarded here if the translations did not occur in our in-house bilingual phrase dictionary. This allowed us to obtain a more important number of unique (q, t) pairs (389 989 after filtering spurious transpots).

The reduction of the variants to display for each query was quantitatively evaluated with two versions of our merge process, which differ only in the edit

distance used to compare various forms. The first version merges only transpots that are inflected forms of the same lemmatized sequence or that only varies by punctuation marks; it leads to a significant decrease in the number of forms to display since we get 1.22 variants per pattern. The second version merges not only the transpots that would be clustered by the previous one, but also variants of word sequences that differ by the use of grammatical words; this method results in a higher number of variants per pattern (1.88 on average).

From these numbers, it can be seen that merging grammatical words dramatically decreases the number of outputs of our system, thereby allowing for the display of more different translations. It often leads to patterns that are easily understandable. For example, our system merges the sequences `manière générale`, `de manière générale`, `une manière générale` and `d'une manière générale` into the pattern `[de]? (une)? manière générale` where `(.)?` or `[.]?` notes optional words and `[`w`]` indicates that w is a word lemmatized from several merged words inflected from w. Our algorithm also builds patterns such as `(avec|sur|durant) (des|les|plusieurs) années` that succeed in grouping similar variants.

Sometimes, merging grammatical words generates patterns from expressions that are not synonymous, which requires a refinement of our method. For instance, `(tout)? [faire] (tout)? (ce)?` was built, whereas two different patterns would have been better: `tout [faire]` and `[faire] tout (ce)?`. In another example, the pattern `(qui)? (s')? (en|à)? [venir]` made from the variants `à venir`, `qui vient`, `qui viennent` or `qui s'en viennent` is difficult to understand. We are counting on the input of end-users to help us decide on the optimum manner of displaying variant translations.

5 Discussion

In this study, we have investigated the use of statistical word-alignment within `TransSearch`, a bilingual concordancer. Overall, we found that our best transpotting algorithm ContigVit-spotter, a Viterbi aligner with a contiguity constraint, combined with a filter to remove spurious transpots, significantly outperforms other transpotting methods, with a F-measure of 0.470. We have demonstrated that it is possible to detect erroneous transpots better than a fair baseline, and that merging variants of a prototypical translation can be done efficiently.

For the time being, it is difficult to compare our results to others in the community. This is principally due to the uniqueness of the `TransSearch` system, which archives a huge translation memory. To give a point of comparison, in [13] the authors report alignment results they obtained for 120 selected queries and a TM of 50 000 pairs of sentences. This is several orders of magnitude smaller than the experiments we conducted in this study.

There are several issues we are currently investigating. First, we only considered simple word-alignment models in this study. Higher-level IBM models can potentially improve the quality of the word alignments produced. At the

very least, HMM models [14], for which Viterbi alignments can be computed efficiently, should be considered. The alignment method used in current phrase-based SMT is another alternative we are considering.

Acknowledgements

This research is being funded by an NSERC grant. The authors wish to thank Elliott Macklovitch for his contribution to this work.

References

1. Macklovitch, E., Lapalme, G., Gotti, F.: Transsearch: What are translators looking for? In: 18th Conference of the Association for Machine Translation in the Americas (AMTA), Waikiki, Hawai'i, USA (2008) 412–419
2. Simard, M., Macklovitch, E.: Studying the human translation process through the TransSearch log-files. In: AAAI Symposium on "Knowledge Collection from volunteer contributors", Stanford, CA, USA (2005)
3. Véronis, J., Langlais, P.: 19. In: Evaluation of Parallel text Alignment Systems — The Arcade Project. Kluwer Academic Publisher, Dordrecht, The Netherlands (2000) 369–388
4. Simard, M.: Translation spotting for translation memories. In: HLT-NAACL 2003 Workshop on Building and using parallel texts: data driven machine translation and beyond, Edmonton, Canada (2003) 65–72
5. Brown, P., Della Pietra, V., Della Pietra, S., Mercer, R.: The mathematics of statistical machine translation: parameter estimation. Computational Linguistics **19**(2) (1993) 263–311
6. Freund, Y., Schapire, R.: Large margin classification using the perceptron algorithm. Machine Learning **37**(3) (1999) 277–296
7. Collins, M.: Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In: EMNLP 2002, Philadelphia, PA, USA (2002) 1–8
8. Liang, P., Bouchard-Côté, A., Klein, D., Taskar, B.: An end-to-end discriminative approach to machine translation. In: 21st COLING and 44th ACL, Sydney, Australia (2006) 761–768
9. Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T.J., Higgins, D.G., Thompson, J.D.: Multiple sequence alignment with the Clustal series of programs. Nucleic Acids Research **31**(13) (2003) 3497–3500
10. Saiou, N., Nei, M.: The neighbor-joining method: A new method for reconstructing phylogenetic trees. Molecular Biology and Evolution **4**(4) (1987) 406–425
11. Och, F.J., Ney, H.: A systematic comparison of various statistical alignment models. Computational Linguistics **29**(1) (2003) 19–51
12. Fleiss, J.L., Levin, B., Pai, M.C.: Statistical Methods for Rates and Proportions. 3rd edn. Wiley-Interscience (2003)
13. Callisson-Burch, C., Bannard, C., Schroeder, J.: A compact data structure for searchable translation memories. In: 10th European Conference of the Association for Machine Translation (EAMT), Budapest, Hungary (2005) 59–65
14. Vogel, S., Ney, H., C., T.: HMM-based word alignment in statistical translation. In: 16th conference on Computational linguistics, Copenhagen, Denmark (1996) 836–841