1

Towards Efficient Translation Memory Search Based on Multiple Sentence Signatures

Juan M. Huerta
IBM TJ Watson Research Center,
United States

1. Introduction

The goal of machine translation is to translate a sentence *S* originally generated in a source language into a sentence *T* in target language. Traditionally in machine translation (and in particular in Statistical Machine Translation) large parallel corpora are gathered and used to create inventories of sub-sentenial units and these, in turn, are combined to create the hypothesis sentence *T* in the target language that has the maximum likelihood given S. This approach is very flexible as it has the advantage generating reasonable hypotheses even when the input has not resemblance with the training data. However, the most significant disadvantage of Machine Translation is the risk of generating sentences with unnaceptable linguistic (i.e., syntactic, grammatical or pragmatic) incosistences and imprefections.

Because of this potential problem and because of the availability of large parallel corpora, MT researchers have recently begun to expore the *direct* search approach using these translation databases in support of Machine Translation. In these approaches, the underlying assumption is that if an input sentence (which we call a *query*) *S* is sufficiently similar to a previously hand translated sentence in the memory, it is, in general, preferable to use such existing translations over the generated Machine Translation hypothesis. For this approach to be practical there needs to exist a sufficiently large database, and it should be possible to identify and retrieve this translation in in a span of time comparable to what it takes for the Machine Translation engine to carry out its task. Hence, the need of algorithms to efficiently search these large databases.

In this work we focus on a novel approach to Machine Translation memory lookup based on the efficient and incremental computation of the string edit distance. The string edit distance (SED) between two strings is defined as the number of operations (i.e., insertions, deletions and substitutions) that need to be applied on one string in order to transform it into the second one (Wagner & Fischer, 1974). The SED is a symmetric operation.

To be more precise, our approach leverages the rapid elimination of unpromising candidates using increasingly stringent elimination criteria. Our approach guarantees an optimal answer as long as this answer has an SED from the query smaller than a user defined threshold. In the next section we first introduct string similarity translation memory search, specifically based on the string edit distance computation, and following we present our approach which focuses on speeding up the translation memory search using increasingly stringent sentence signatures. We then describe how to implement our approach using a Map/Reduce framework and we conclude with experiments that illustrate the advantages of our method.

2. Translation memory search based on string similarity

A translation memory consists of a large database of pre-translated sentence pairs. Because these translation memories are typically collections of high quality hand-translations developed for corpus building purposes (or in other cases, created for the internationalization of documentation or for other similar development purpoes), if a sentence to be translated is found *exactly* in a translation memory, or within a relativelly small edit distance from the query, this translation is preferred over the output generated by a Machine Translation engine. In general, because of the nature of the errors introduced by SMT a Translation Memory match with an edit distance smaller than the equivalent average BLEU score of a SMT hypothesis is preferred. To better understand the relationship between equivalend BLEU and SED the reader can refer to (Lin & Och, 2004).

Thus, for a translation memory to be useful in a particular practical domain or application, 3 conditions need to be satisfied:

- It is necessary that human translations are of at least the same average quality (or better) than the equivalemtn machine translation output
- It is necessary that there is at least some overlap between translation memory and the query sentences and
- It is necessary that the translation memory search process is not much more computationally expensive than machine translation

The first assumption is typically true for the state of the current technology and certainly the case when the translations are performed professional translators. The second condition depends not only on the semantic overlap between the memory and the queries but also on other factors such as the sentence length distribution: longer sentences have higher chances of producing matches with larger string edit distances nullifying their usefulness. Also, this condition is more common in certain domains (for example technical document translation where common technical processes lead to the repeated use of similar sentences). The third assumption, (searching in a database of tens of millions of sentences) is not computationally trivial especially since the response time of a typical current server-based Machine Translation engine is of about a few hundred words per second. This third requirement is the main focus of this work.

We are not only interested in finding exact matches but also in finding *high-similarity* matches. Thus, the trivial approach to tackle this problem is to compute the string edit distance between the sentence to be translated (the source sentence) and *all* of the sentences in the database. It is easy to see how when comparing 2 sentences each with length *n* and using Dynamic Programming based String Edit Distance (Navarro, 2001) the number of operations required is $O(n^2)$. Hence, to find the best match in a memory with *m* senteces the complexity of this approach is $O(mn^2)$. It is easy to see how a domain where a database contains tens of millions of translated sentences and where the average string length is about 10, the naive search approach will need to perform in the order of billions of operations *per query.* Clearly, this naive approach is computationally innefficient.

Approximate string search can be carried out more efficiently. There is a large body of work on efficient *approximate* string matching techniques. In (Navarro, 2001), there is a very extensive overview of the area of approximate string matching approaches. Essentially, there are two types of approximate string similarity search algorithms: on-line and off-line. In the on-line methods, no preprocessing is allowed (i.e., no index of the database is built). In off-line methods an index is allowed to be built prior to search.

We now provide, as background, an overview of existing methods for approximate string match as well as advantages and disadvantages of these in order to position our approach in this context.

2.1 Off-line string similarity search: index based techniques

Off line string similarity approaches typically have the advantage of creating an index of the corpus prior to search (see for example (Bertino et al., 1997). These approaches can search, in this way, for matches much faster. In these methods terms can be weighted by their individual capability to contribute to the overall recall of documents (or, in this case, sentences) such as TD-IDF or Okapi BM25.

However, index-based approaches are typically based on so called bag-of-words distance computation in which the sentences are converted into vectors representing only word count values. Mainly because of their inability to model word position information, these approaches can only provide a result without any guarantee of optimality. In other words, the best match returned by an index query might not contain the best scoring sentence in the database given the query. Among the reasons that contribute to this behavior are: the non-linear weights of the terms (e.g., TF-IDF), the possible omission of stop words, and primarily the out-of-order nature of the bag of words approach.

To overcome this problem, approaches based on positional indexes have been proposed (Manning, 2008). While these indexes are better able to render the correct answer, they do so at the expense of a much larger index and a considerable increase in computational complexity. The complexity of a search using a positional index is *O(T)* where T denotes the number of tokens in the memory *(T=nm)*. Other methods combine various index types like positional, next word and bi-word indices (e.g., (Williams et al., 2004)). Again, in these cases accuracy is attained at the expense of computational complexity.

2.2 On-line string similarity matching: string edit distance techniques

There are multiple approaches to on-line approximate string matching. These, as we said, do not benefit from an index built *a-priori*. Some of these approaches are intended to search for exact matches only (and not approximate matches). Examples of on line methods include methods based on Tries (Wang et al., 2010) (Navarro & Baeza-Yates, 2001), finite state machines, etc. For an excellent overview in the subject see (Navarro, 2001).

Our particular problem, translation memory search, requires the computation of the string edit distance between a candidate sentence (a query) and a large collection of sentences (the translation memory). Because as we saw in the previous section the string edit distance is an operation that is generally expensive to compute with large databases, there exist alternatives to the quick computation of the string edit distance. In order to better explain our approach we first start by describing the basic dynamic programming approach to string edit distance computation.

Consider the problem of computing the string edit distance between two strings *A* and *B*. Let $A=\{a_1,...a_n\}$ and $B=\{b_1,...b_m\}$. The dynamic programming algorithm, as explained in (Needleham & Wunsch, 1970), consists of computing a matrix *D* of dimensions *(m+1)x(n+1)* called the edit-distance-matrix, where the entry *D[i,j]* is the edit distance *SED(Ai,Bj)* between the prefixes A_i and B_j. The fundamental dynamic programming recurrence is thus,

$$D[i,j] = \min \left\{ \begin{array}{c} D[i-1,j]+1 \quad \text{if } i>0 \\ D[i,j-1]+1 \quad \text{if } j>0 \\ D[i-1,j-1]+\partial_{a,b} \quad \text{if } i>0 j>0 \end{array} \right\} \tag{1.1}$$

The initial condition is *D[0,0]=0*. The edit distance between *A* and *B* is found in the lower right cell in the matrix *D[m,n]*. We can see that the computation of the Dynamic Programming can be carried out in practice by filling out the columns (j) of the DP array. Figure 1 below, shows the DP matrix between sentences Sentence_1="A B C A A" and Sentence_2="D C C A C" (for simplicity words are considered in this example to be letters). We can see that the distance between these two sentences is 3, and the cells in bold are the cells that constitute the optimal alignment path.

2.2.1 Sentence pair improvements on methods based on the DP matrix

A taxonomy of approximate string matching algorithms based on the dynamic programming matrix is provided in (Navarro, 2001). Out of this taxonomy, two approaches are particularly relevant to our work. The first is the approach of Landau Vishkin 89, which focusing on a Diagonal Transition manages to reduce the computation time to *O(kn)* where *k* is the maximum number of expected errors *(k<n)*. The second is Ukkonen 85b which based on a cutoff heuristic also reduces the computation time to *O(kn)*. Our work is based on a multi-signature approach that uses ideas similar to the heuristics of Ukkonen.

		i	0	1	2	3	4	5
i				**A**	**B**	**C**	**A**	**A**
0			**0**	1	2	3	4	5
1	**D**		1	**1**	2	3	4	5
2	**C**		2	2	**2**	2	4	5
3	**C**		3	3	3	**2**	3	4
4	**A**		4	3	4	3	**2**	3
5	**C**		5	4	4	4	3	**3**

Fig. 1. Sample Dynamic Programming Matrix

2.2.2 Approximate string edit distance computation

In section 2.1 we described an off-line method for segment retrieval based on an index and a bag of words approach. That approach does not intend to approximate the string edit distance; rather, it calculates similarity based on term distribution vectors and thus produces results that differ from the SED approach. To reduce this mismatch between off-line and on-line methods, it is possible to approximate the SED (and the related Longest Common Subsequence computation) based on a stack computation and information derived from a positional index. This computation is possible through the use of a Stack structure and a A* like algorithm as described in (Huerta, 2010b). In that paper, Huerta proposed a method that takes *O(m s log s)* operations on average where s is the depth of the stack (typically much smaller than *T*, or *m*) instead of *O(T)* using a positional index. This approach is important because it improves the accuracy of an off line system using a position index by using an approximation of the string edit distance without sacrificing speed. The results are within 2.5% error (Huerta, 2010b). In this paper, we will focus exclusively on the on-line approach.

3. Multi-signature SED computation

Our approach is intended to produce the best possible results (i.e., find the best match in a translation memory given a query if this exists within a certain *k*, or number of edits) at speeds comparable to those produced by less accurate approaches (i.e., indexing), in a way that is efficiently parallelizable (specifically, implementable in Map Reduce). To achieve this, our approach decomposes the typical single DP computation into multiple consecutive string signature based computations in which candidate sentences are rapidly eliminated. Each signature computation is much faster than any of its subsequent computations.

The core idea of the signature approach is to define a hypersphere of radius equal to *k* in which to carry out the search. In other words, a cutoff is used to discard hypotheses. Eventually the hypersphere can be empty (without hypotheses) if there is no single match within the cutoff (whose distance is smaller than the cutoff).

The idea is that, at each signature stage the algorithm should be able to decide efficiently with certainty if a sentence lies outside the hypersphere. By starting each stage with a very large number of candidates and eliminating a subset, the algorithm shows the equivalent of perfect recall but its precision only increases with a rate inversely proportional to the running speed. The signature based algorithms (the kernels) are designed to be very fast at detecting out of bound sentences and slower at carrying out exact score computations. We start by describing the 3 signature based computations of our approach.

3.1 Signature 1: string length

The first signature computation is carried out taking into account the length of the query string as well as the length of the candidate string. This first step is designed to eliminate a large percentage of the possible candidates in the memory very rapidly. The idea is very simple: a pair of strings S1 and S2 cannot be within *k* edits if $|l1\text{-}l2|$>k, where *l1* is the length of string 1 and so on.

Figure 1 below shows a histogram of the distribution of the length of a translation memory consisting of 9.89 Million sentence pairs. As we can see, the peak is at sentences of length 9 and consists of 442914 sentences which correspond to about 4.5% of the sentences. But the average length is 14.5 with a standard deviation of 9.18, meaning that there is a long tail in the distribution (a few very long sentences). We assume that the distribution of the query strings matches that of the memory. The *worst case*, for the particular case of *k*=2, constitutes the bins in the range $l1\text{-}k<l2<l1+l2$. This in our case is the case of the query equals to 9. For this particular case and memory the search space is reduced to 2.18M (i.e., to 22% and hence reduced by 78%). This is in the worst case that happens only 4.5% of the time. The weighted average improvement is a reduction of the search space to 10% of the original. This, in turn speeds up search on average 10x.

An even faster elimination of candidates is possible if multiple values of *k* are used depending on the length of the memory hypotheses. For example one can run with a standard k for hypotheses larger than 10 and a smaller k for hypotheses smaller or equal to 10. One can see from the distribution of the data that the overall result of this first signature step is the elimination of between 70% to more than 90% of the possible candidates, depending on the specific memory distribution.

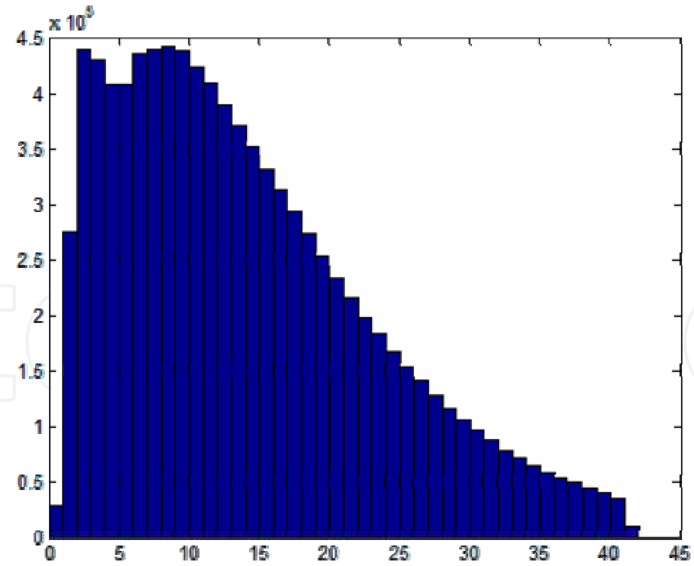


Fig. 2. Histogram of distribution of sentence lengths for a Translation Memory

3.2 Signature 2: lexical distribution signature

The second signature operation is related to the lexical frequency distribution and consists of a fast match computation based on the particular words of the query and the candidate sentences. We leverage the Zipf-like distribution (Ha et al., 2001) of the occurrence frequency of words in the memory and the query to speed up this comparison. To carry out this operation we compute the sentence lexical signature, which for Sentence *Si* is a vector of length *li* consisting of the words in the sentence sorted by decreasing rarity (i.e., increasing frequency). We describe in this section an approach to detect up to k-differences in a pair of sentence signatures in time (worst case) less than *O(n)* (where n is the sentence average length). The algorithm (shown in figure 3 below) stops as soon as *k* differences between the signature of the query and the signature of the hypothesis are observed, and the particular hypothesis is eliminated.

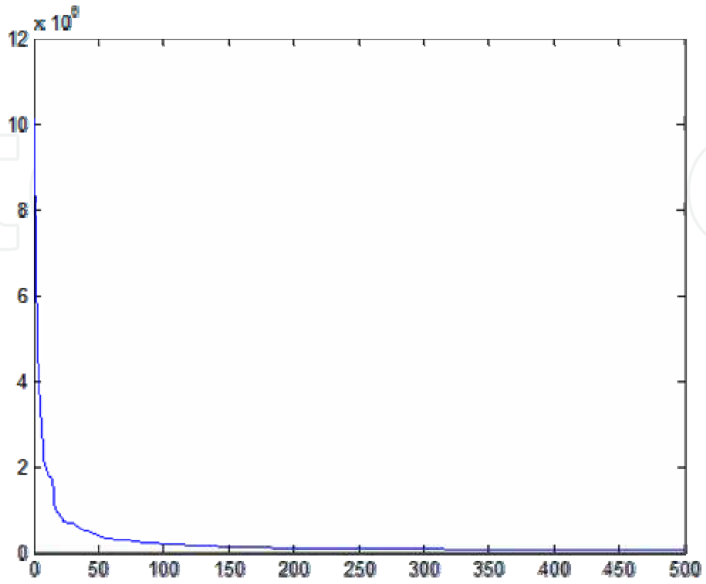


Fig. 3. Frequency distribution of words in the sample Translation Memory

To better motivate our algorithm let us explain a little bit the distribution of the words in the translation memory we use as an example. First, we address the question, how unique is the lexical signature of a sentence? Empirically, we can see in our sample translation memory that out of the 9.89M sentences, there are 9.85M unique lexical signatures, meaning that this information by itself could constitute a good match indicator. Less than 1.0% of the sentences in the corpus share a lexical signature with another sentence. As we said, the signature is based on the Zipf's distribution of word frequencies. It has been observed (Ha et al) that at least for the case of the most frequent words in a language is inversely proportional to rank.

We now describe the algorithm to efficiently compute the bound in differences in lexical items between two sentences.

```
Search Algorithm
construct the sorted vector of instances for A and B. Sa and Sb
i=0,j=0, d=0;
while number of differences is less than k  (d<k)
    if Sa[i]==Sb[j]
            then
             i++; j++; break;
     else if Sa[i]>Sb[j]
           d++;
           j++;
    else
          d++;
         i++;
end while
```

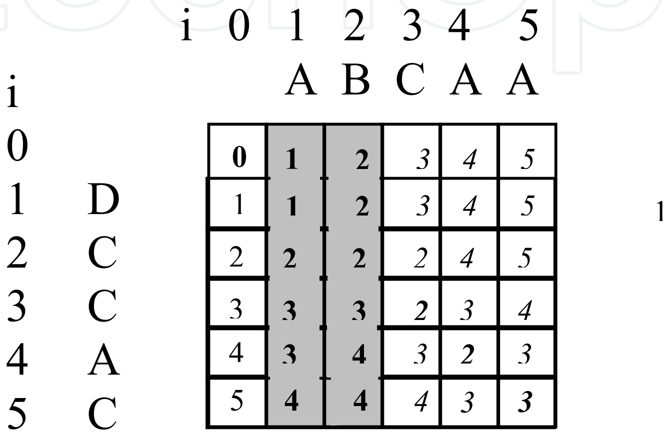
Fig. 4. Search Algorithm for Lexical Frequency Distribution String Signature

It is possible to show that the above code will stop (quit the *while* loop) on an average proportional to *O(alpha k)* where *alpha* is bounded by characteristics of the word frequency distribution. Also, the worst case is *O(n)* (when the source is equal to the target) this will happen with an empiric probability of less than 0.01 in a corpus like the one we use. The best case is *O(k)*.

As we will see in later sections, this signature approach combined with Map-Reduce produces very efficient implementations: the final kernel performs an exact computation passing from the Map to the Reduce steps only those hypotheses within the radius. The Reduce step finds the best match for every given query by computing the DP matrix for each query/hypothesis pair. The speedup in this approach is proportional to the ratio of the volume enclosed in the sphere divided by the whole volume of the search space.

3.3 Signature 3: bounded dynamic programming on filtered text

After the first two signature steps, a significant number of candidate hypotheses have been eliminated based on string length and lexical content. In the third step a modified Dynamic Programming computation is performed over all the surviving hypotheses and the query. The matrix based DP computation has two simple differences from the basic algorithm described before. The first one instructs the algorithm to stop after the minimum distance in an alignment is *k* (i.e., when the smallest value in last column is k) and sets its focus on the diagonal. The second modification relates to the interchange of the sentences so that the longest sentence in the columns and the shortest one in the rows. Figure 5 below shows an example of a DP matrix in which in the second column is obvious that the minimum alignment distance is at least 2. If, for this particular case, we were interested in k<2, then the algorithm would need to stop at this point. An additional difference is also possible and further increases the speed: each sentence in the memory (and the query itself) are represented by non-negative integers, where each integer represents a word id based on a dictionary. In our experiments we used very large dictionaries (400k), in which elements such as URL's and other special named entities are all mapped to the *unknown word* ID.



i		0	1	2	3	4	5
			A	B	C	A	A
0		0	1	2	3	4	5
1	D	1	1	2	3	4	5
2	C	2	2	2	2	4	5
3	C	3	3	3	2	3	4
4	A	4	3	4	3	2	3
5	C	5	4	4	4	3	3

Fig. 5. Bounded DP Matrix

3.4 Combining signatures: representation of the memory

We have described how to carry out the multi-signature algorithm. While this approach significantly increases the search speed, for it to be truly efficient in practice, it should avoid computing the signature information of the translation memory for each query it receives. Rather it should use a slightly bigger pre-computed data structure in which for each sentence, the length signature and the lexical signature are available.

The translation memory will thus consist of one record for each sentence in the memory. Each record in the memory will consist of the following fields: The first field has the sentence length. The second field has the lexical signature vector for a sentence. The third field has the dictionary filtered memory sentence. The fourth field has the plain text sentence. While this representation increases the size of the memory by a factor of at least 3, we have found that it is extremely useful in keeping the efficiency of the algorithm.

4. Map/Reduce parallelization

We previously described that our multi-signature algorithm can be further sped-up by carrying it out in a parallelized fashion. In this section we describe how to do so based on the Map/Reduce formulation, specifically on Hadoop.

Map-Reduce (and in particular Hadoop) (Dean & Ghemawat, 2008) is a very useful framework to support distributed in large computer clusters. The basic idea is to segment the large dataset (in our case, the memory) and provide portions of this partition to each of the worker nodes in the computing cluster. The worker nodes perform some operation on the segment of the partition domain and provides results in a data structure (a hash map) consisting of key-value pairs. All the output produced by the worker nodes is collated and post-processed in the Reduce step, where a final result set is obtained. An illustration of the general process is shown in figure 6.

In our particular Map Reduce implementation the translation memory constitutes the input records that are partitioned and delivered to the map worker nodes. Each Map job reads the file with the translation queries and associates each with a key. Using the multi signature approach described above, each worker node rapidly evaluates the SED-feasibility of candidate memory entries and, for those whose score lies within a certain cutoff, it creates an entry in the hash map. This entry has as the key the query sentence id, and as the value a structure with the SED score and memory id entry.

In the reduce step, for every query sentence, all the entries whose key correspond to the particular query sentence in question are collated and the best candidate (or top N-best) are selected. Possibly, this set can be empty for a particular sentence if no hypotheses existed in the memory within k-edits.

It is easy to see that if the memory has *m* records and the query set has *q* queries the maximum set of map records is *qm*. Hadoop sorts and collates these map records prior to the reduce step. Thus in a job where the memory has 10M records and the query set ha 10k sentences, the number of records to sort and to collate are 100Billion. This is a significantly large collection of data to organize and sort. It is crucial to reduce this number and as we cannot reduce *q* our only alternative is to reduce *m*. That is precisely the motivation behind the multi-signature approach. The multi-signature approach that is proposed in this work not only avoids the creation of such large number of Map records but also reduces the exact Dynamic Programming computation spent in the Map jobs.

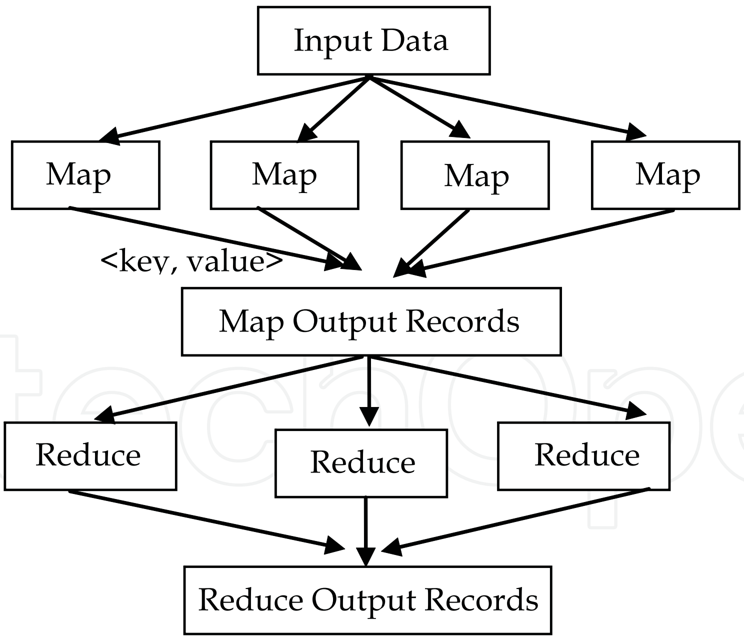


Fig. 6. Map Reduce Diagram

5. Experiments

To conclude, this section binds all the previous themes by providing a set of experiments describing the exact string similarity computation speed and coverage results. To test our algorithms we explored the use of an English-to-Spanish Translation memory consisting of over 10M translation pairs.

Our test query sentences consist of a set of 7795 sentences, comprising 125k tokens (words). That means the average query length is 16.0 words per sentence. Figure 7 shows the histogram of the distribution of the lengths in the query set. We can see in this histogram that there are 2 modes: one is for sentences of length 3 and the other is for sentences of length 20.

Our Hadoop environment for Map Reduce parallelization consists of 5 servers: 1 name node and 4 dedicated to data node servers (processing). The name nodes have 4 cores per CPU resulting in a total of 16 processing cores. We partitioned the memories in 16 parts and carried out 16 map tasks, 16 combine tasks (which consolidate the output of the map step prior to the reduce task) and 1 reduce task. The file system was an HDFS (Hadoop Distributed File System) consisting on one Hard Drive per server (for a total of 5 Hard Drives).

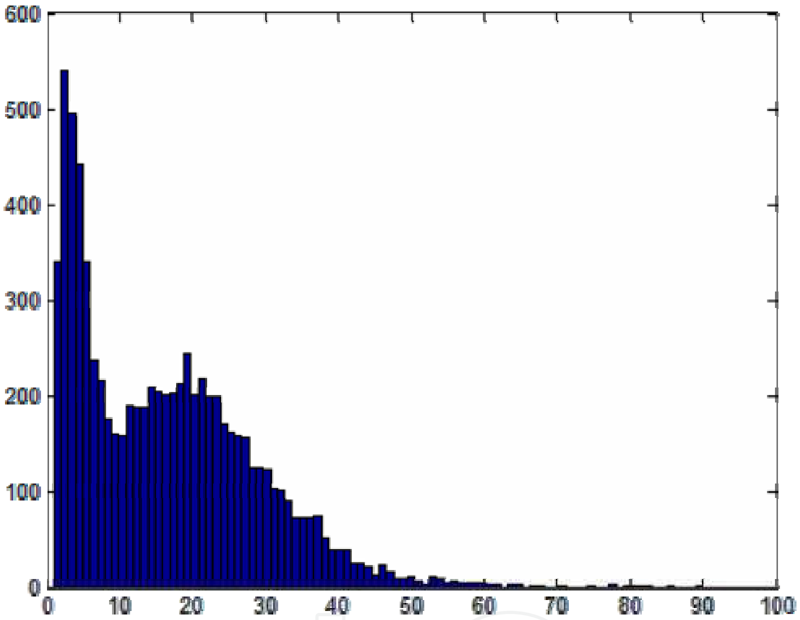


Fig. 7. Histogram of distribution of sentence lengths for a Query Set

We ran two sets of experiments using various cutoff configurations. In the first set of configurations, the map-reduce jobs use a single cutoff that is applied equally to all the query sentences and hypotheses. In the second configuration for each sentence apply one of 2 cutoffs depending on the length of each query.

Table 1 shows the results for the case of the single cutoff (Cutoff 1= Cutoff 2). Column 1 and 2 correspond to the cutoff (which is the same), in Column 3 we can see the number of queries found, Column 4 shows the total time it took to complete the batch of queries (in seconds), and Column 5 shows the total number of Map records. As we can see the as we increase the cutoff the number of output sentences, the time and the number of map records increase. We will discuss in more detail the relationship between these columns. Table 2 shows the same results but in this case Cutoff 1 (for short queries) is not necessarily equal to Cutoff 2 (for long queries).

Cutoff 1	Cutoff 2	Output Sentences	Time (s)	Map Records
1	1	773	148	773
2	2	1727	170	10.24M
3	3	2355	411	277M
4	4	2749	898	1.04B
5	5	3056	1305	2.048B
6	6	3285	2145	3.14B

Table 1. Experimental results for Cutoff1=Cutoff2

Cutoff 1	Cutoff 2	Output Sentences	Time (s)	Map Records
2	5	2770	271	20.4M
2	6	2999	475	344M
2	7	3271	581	716M

Table 2. Experimental results for length specific Cutoff

We can see that if we are allowed to have two length-related cutoffs, the resulting number of output sentences is kept high at a much faster response time (measured in seconds per query). So for example if one wanted to obtain 2700 sentences we can use cutoff 2 and 5 and run in 271 seconds or alternatively use 4 and 4 and run in 898 seconds. The typical configuration attains 2770 sentences (cutoffs 2 and 5) in 271 seconds for an input of size 7795 which means 34 ms per query. This response time is typical, or faster that a translation engine and this allows for our approach to be a feasible runtime technique.

Figure 8 shows the plot of the total processing time for the whole input set (7795 sentences) as a function of the input cutoff. Interestingly, one can see that the curve follows a non-linear trend as a function of the cutoff. This means that as the cutoff increases, the number of operations carried out by our algorithm increases non-linearly. But, how exactly are these related?

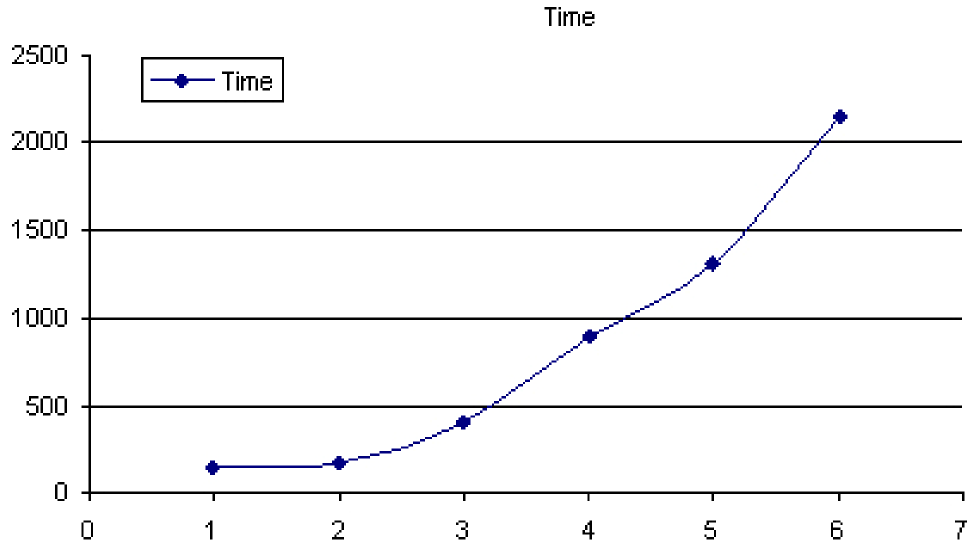


Fig. 8. Time as a function of single cutoff

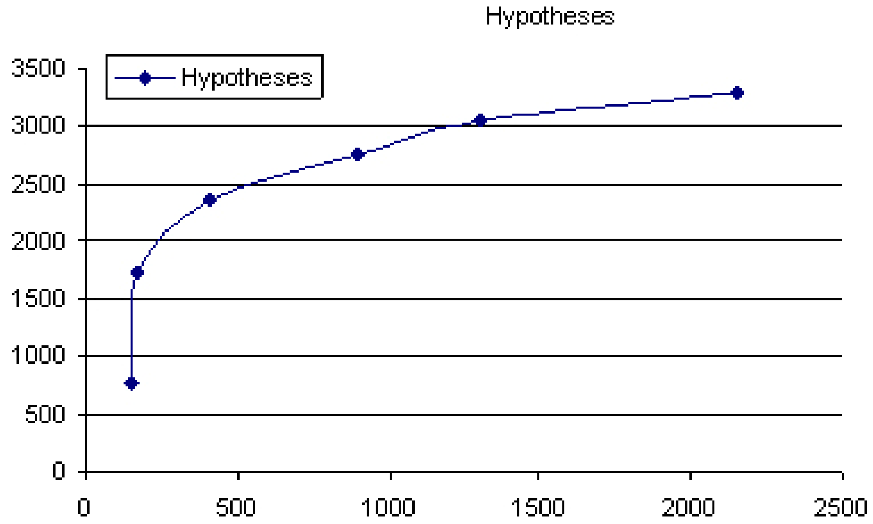


Fig. 9. Number of Output Hypotheses as a Function of Total Run Time

To explore this question, Figure 9 shows the total number of output sentences as a function of total runtime. We see that in the first points there is a large increase in output hypotheses per additional unit of time. After the knee in the curve, though, it seems that the system stagnates as it reduces its output per increment in processing time. This indicates that the growth in processing time that we are observing by increasing the threshold is not the direct result of more output hypotheses being generated. As we will se below, rather it is the result of a growth in processing map records.

In Figure 10 we show the total number of map records (in thousands) as a function of observed total run-time. Interestingly, this is an almost linear function (the linear trend is also shown).As we have mentioned, the goal of our algorithm is to minimize the records it produces. Having minimized the number of records, we have effectively reduced the run-time.

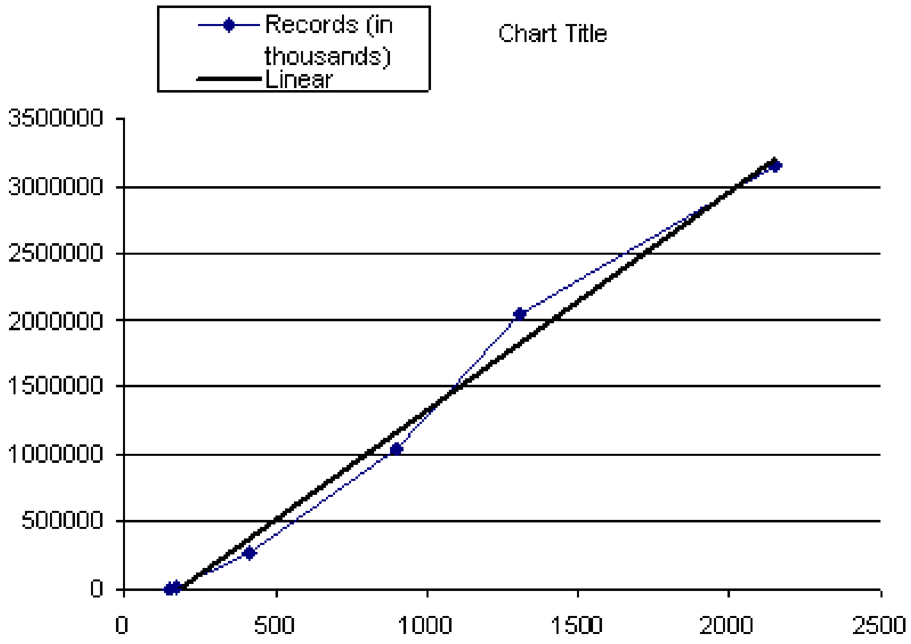


Fig. 10. Number of Map Records (in thousands) as a Function of Total Run Time

Finally Figure 11 shows the number of records per number of output hypotheses. This figure tells us about the efficiency of the system in producing more output. We can see that as the system strives to produce more output matches a substantially larger number of records needs to be considered considerably increasing the computation time.

6. Conclusion

We have presented here an approach to translation memory retrieval based on the efficient search in a translation pair database. This approach leverages several characteristics of natural sentences like the sentence length distribution, the skewed distribution of the word occurrence frequencies as well as DP matrix computation optimization into consecutive sentence signature operations. The use of these signatures allows for a great increase in the efficiency of the search by removing unlikely sentences from the candidate pool. We demonstrated how our approach combines very well with the map reduce approach.

In our results we found how the increase in running time experienced by our algorithm as the cutoff is increased grows in a non-linear way. We saw how this run time is actually related to the total number of records handled by Map Reduce. Therefore it is important to reduce the number of unnecessary records without increasing the time to carry out the signature computation. This is precisely the motivation behind the multi-signature approach described in this work.

The approach described in this paper can be applied directly into other sentence similarity approaches, such as sentence-based Language Modelling based on IR (Huerta, 2011) and others where large textual collections are the norm (like Social Network data, (Huerta, 2010)). Finally, this approach can also be advantageously extended to other non-textual domains in which the problem consists of finding the most similar sequence (e.g., DNA sequences etc) where the symbol frequency distributions of the domain sequences is skewed and there is a relatively broad sequence length distribution.

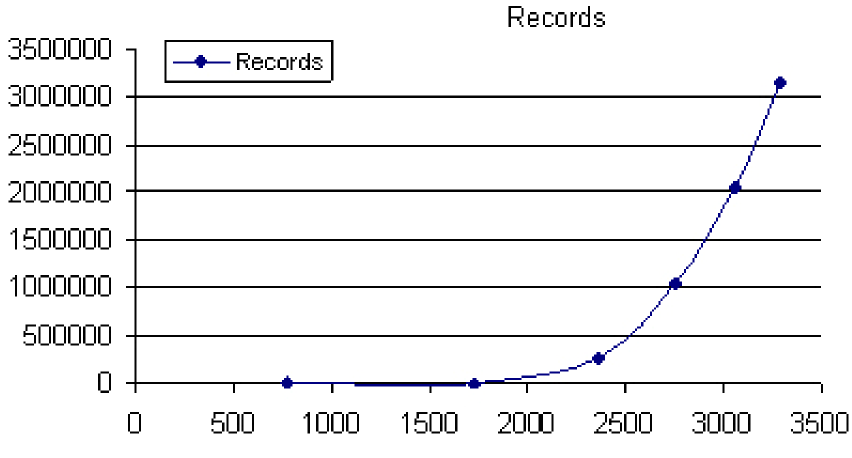


Fig. 11. Number of Map Records as a Function of Total Output Hypotheses

7. References

Bertino E., Tan K.L., Ooi B.C., Sacks-Davis R., Zobel J., & Shidlovsky B. (1997). *Indexing Techniques for Advanced Database Systems*. Kluwer Academic Publishers, Norwell, MA, USA

Dean J., & Ghemawat S. (2008). *MapReduce: simplified data processing on large clusters*. Commun. ACM 51, 1 (January 2008), 107-113

Ha L. Q., Sicilia-Garcia E. I., Ming J., & Smith F. J. (2002). *Extension of Zipf's law to words and phrases*. In Proceedings of the 19th international conference on Computational linguistics - Volume 1 (COLING '02), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 1-6.

Huerta J. M. (2011), *Subsequence Similarity Language Models*, Proc. International Conference in Acoustics Speech and Signal Processing 2011

Huerta J. M. (2010), *An Information-Retrieval Approach to Language Modeling: Applications to Social Data* NAACL #Social Media Workshop

Huerta J. M. (2010b) *A Stack Decoder Approach to Approximate String Matching* In Proc. SIGIR 2010

Landau G. M., Myers E. W., & Schmidt J. P. (1998). Incremental String Comparison. SIAM J. Comput. 27, 2 (April 1998), 557-582.

Lin C. W., & Och F. J. (2004). *Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics*. In Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics ACL '04

Manning C. D., Raghavan P., & Schütze H. (2008). *Introduction to Information Retrieval*, Cambridge University Press. 2008.

Navarro G., Baeza-Yates R., Sutinen E., & Tarhio J. (2001). *Indexing methods for approximate string matching,* IEEE Data Engineering Bulletin, 2001

Navarro G. (2001). *A guided tour to approximate string matching*, ACM Computing Surveys v.33 No. 1 2001

Needleham S.B. & Wunsch C.D. (1970). *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, J. of Mol. Bio. Vol 48 (1970), pp. 443-453.

Wagner R.A. & Fischer M. J. (1974). *The string-to-string correction problem,* J. of the ACM, vol. 21 No. 1 (1974) pp-168-173

Wang, J. Fang J. & Li G. (2010). *TrieJoin:Efficient Triebased String Similarity Joins with EditDistance Constraints* The 36th International Conference on Very Large Data Bases, September 1317,

Williams H.E., Zobel J., & Bahle D. (2004). *Fast Phrase Querying with Combined Indexes*. ACM Transactions on Information Systems, 22(4) , 573-594, 2004

Speech and Language Technologies
Edited by Prof. Ivo Ipsic

ISBN 978-953-307-322-4
Hard cover, 344 pages
Publisher InTech
Published online 21, June, 2011
Published in print edition June, 2011

This book addresses state-of-the-art systems and achievements in various topics in the research field of speech and language technologies. Book chapters are organized in different sections covering diverse problems, which have to be solved in speech recognition and language understanding systems. In the first section machine translation systems based on large parallel corpora using rule-based and statistical-based translation methods are presented. The third chapter presents work on real time two way speech-to-speech translation systems. In the second section two papers explore the use of speech technologies in language learning. The third section presents a work on language modeling used for speech recognition. The chapters in section Text-to-speech systems and emotional speech describe corpus-based speech synthesis and highlight the importance of speech prosody in speech recognition. In the fifth section the problem of speaker diarization is addressed. The last section presents various topics in speech technology applications like audio-visual speech recognition and lip reading systems.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821