# Improving fuzzy matching through syntactic knowledge

Authors: Tom Vanallemeersch, Vincent Vandeghinste

Affiliation: Centre for Computational Linguistics, University of Leuven

## Abstract

Fuzzy matching in translation memories (TM) is mostly string-based in current CAT tools. These tools look for TM sentences highly similar to an input sentence, using edit distance to detect the differences between sentences. Current CAT tools use limited or no linguistic knowledge in this procedure. In the recently started SCATE project, which aims at improving translators' efficiency, we apply syntactic fuzzy matching in order to detect abstract similarities and to increase the number of fuzzy matches. We parse TM sentences in order to create hierarchical structures identifying constituents and/or dependencies. We calculate TER (Translation Error Rate) between an existing human translation of an input sentence and the translation of its fuzzy match in TM. This allows us to assess the usefulness of syntactic matching with respect to string-based matching. First results hint at the potential of syntactic matching to lower TER rates for sentences with a low match score in a string-based setting.

## Acknowledgments

This research is funded by the Flemish government agency IWT[1] (project 130041, *SCATE*).

## 1. Introduction

Computer-aided translation (CAT) has become an essential aspect of translators' working environments. CAT tools speed up translation work, create more consistent translations, and reduce repetitiveness of the translation work. One of the core components of a CAT tool is the translation memory system (TMS). It contains a database of already translated fragments, called the translation memory (TM), which consists of translation units: segments of texts (sentences, titles, cell tables, etc.) together with their translation. Given a sentence to be translated (which we will call the *query sentence)*, the CAT tool looks for source language sentences in a TM which are identical (exact matches) or highly similar (fuzzy matches), and, upon success, suggests the translation of the matching sentence to the translator.

In current CAT tools, techniques for retrieving fuzzy matches from a TM mainly consider sentences as simple sequences of words[2] and contain very limited linguistic knowledge, for instance in the form of stop word lists. Few tools use more elaborate syntactic knowledge.[3] In

[1] Innovation by Science and Technology

[2] Words are generally defined as sequences of characters surrounded by spaces.

[3] One example is the tool Similis (http://www.similis.org), which determines constituents such as noun phrases in sentences and allows for retrieving TM sentences which share constituents with the query sentence.

the SCATE project (Smart Computer Aided Translation Environment),[4] which primarily aims at improving translators' efficiency and consistency, we study the use of syntactic information for detecting TM sentences which are not only similar when comparing words but also when comparing the syntactic information associated with the sentences (such information consists for instance of parse trees with constituents and part-of-speech tags). We investigate whether such abstract, syntax-based matching is able to increase the number of matches (recall) and produce matches which lead to more useful translation suggestions. As the SCATE project is a collaboration between university teams and companies such as translation agencies, which provide feedback on techniques developed within the project and on their potential application in an industrial environment, we do not neglect the aspect of matching speed. This is for instance relevant in case of syntax-based matching methods like comparison of parse trees.

The remainder of this paper is structured as follows. In the next section, we provide background on matching methods which are based on words or syntactic information, and on how to evaluate their results. In the subsequent section, we detail our methodology, i.e. the matching and evaluation methods we applied. We then proceed to present the results from our first, exploratory tests, and finally provide conclusions and a lookout to future research.

## 2. Background

There is a great variety of methods for matching flat sequences of elements and for matching complex structures like trees. Flat sequences may consists of various elements, like words or characters (one may compare a pair of words based on their characters). Trees are structures with hierarchically linked nodes. For instance, parse trees structure a sentence into its syntactic constituents, assign part-of-speech tags to words, and assign dependency relations to constituents (for instance *subject*).  An example of a parse tree is shown in Figure 1.

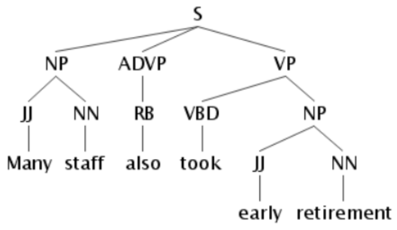

Figure 1: parse tree with syntactic constituents and part-of-speech tags

## 2.1 Strings

Methods acting upon flat sequences are called *string-based*. They include methods like Levenshtein distance (Levenshtein 1966) and percent match (Bloodgood and Strauss 2014). Levenshtein distance is one of the most commonly applied matching methods. It looks for the shortest path to transform one sequence into another sequence through deletions, insertions and substitutions. This shortest path is expressed as a distance. The distance can be converted into a score by normalizing on the sentence length. Although commercial TMS developers do not tend to provide details on their matching methods, Levenshtein distance on sequences of words

[4] http://www.ccl.kuleuven.be/scate

is widely believed to be the major approach (Bloodgood and Strauss 2014), and is also referred to in publicly available system descriptions (Eisele and Lavecchia 2011, Koehn and Senellart 2010a). Table 1 shows an example of Levenshtein distance calculation.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | total | of | 470 | workers | had | accepted | early | retirement |
| Many | staff | also | took | | | | early | retirement |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

→ 4 substitutions +
3 deletions =
cost 7

Table 1: Levenshtein distance

## 2.2. Trees

*Tree-based* methods include tree edit distance (Bille 2005) and tree alignment distance (Jiang et al. 1995). These are two methods with a mathematical rather than linguistic origin. The first method looks for the shortest path to convert one tree to another one by deleting, inserting, and substituting nodes. The second method looks for the easiest way to combine the two trees into a single one by adding empty nodes. When applying these methods to parse trees, node comparison may involve several features on a node, such as word form, lemma (for instance, the infinitive of a verb, the singular form of a noun), part-of-speech tag (for instance, verb) or constituent (for instance, noun phrase). Tree-based methods have also been proposed in the literature on example-based machine translation, see for instance Cromieres and Kurohashi (2011). Figure 2 provides an example of a parse tree alignment based on the shortest tree alignment distance. In this example, the parse trees have a stronger match than the sentences in Table 1, thanks to the fact that the trees contain similar syntactic information: not only some words match, but also some constituent labels and part-of-speech tags.[5]

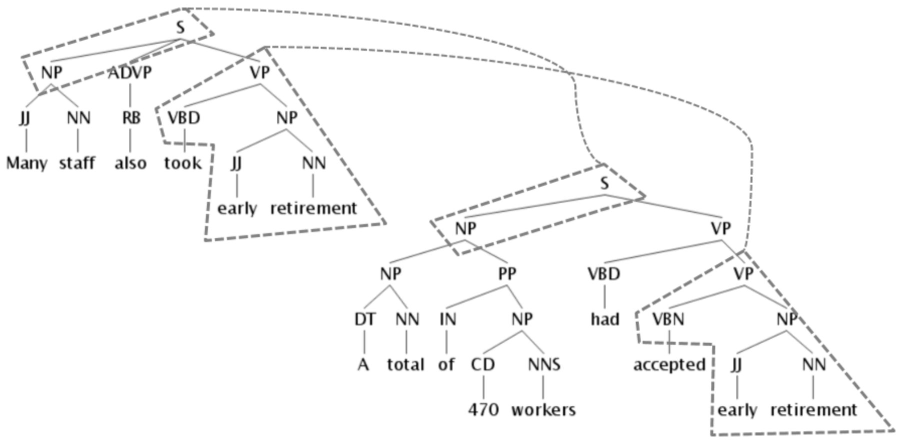

Figure 2: aligned parse trees

[5] The VBD node matches the VBN node as the tags both refer to verbs and the surrounding nodes also match.

## 2.3 Trees as strings

Linguistic knowledge can be exploited in both string-based and tree-based methods. The most obvious matching configuration in this context is applying tree edit distance or tree alignment distance to parse trees. However, it is also possible to use linguistic knowledge in string-based methods. For instance, flat sequences need not consist of word forms, but may also contain word lemmas, part-of-speech tags, etc. Lexical knowledge may be exploited by matching words in flat sequences through lexical similarity. For instance, Gupta (2014) uses a paraphrase database.

Considering tree-based matching methods, the aspect of speed should not be neglected. While methods like Levenshtein distance are highly efficient, even for long sentences, methods like tree alignment distance perform a high number of calculations on a pair of trees. One way to diminish the problem of speed is to convert trees to flat sequences in order to be able to apply string-based methods to them. One such method (Li et al. 2008) is based on work by Prüfer (1918 !). It converts a tree to a flat *Prüfer sequence* by visiting the nodes in the tree in a certain order. Prüfer sequences capture some of the hierarchical information in the tree. Figure 3 illustrates this. Matching on Prüfer sequences helps to select matches in an efficient way before applying a more finegrained and time-consuming method like tree alignment distance.

Another way to tackle the aspect of speed is the use of an index, which allows to reduce the set of source sentences in the TM which are candidate for fuzzy matching methods. One example of such an index is a suffix array; see Koehn and Senellart (2010b). For the sake of briefness, we will not go into further detail on the creation of indices.

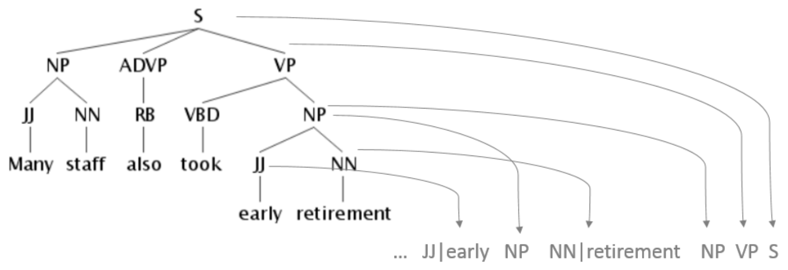

Figure 3: conversion of parse tree to Prüfer sequence

## 2.4 Metrics

While the usefulness of a translation suggestion should ultimately be determined by the user of a CAT tool, human evaluation is time-consuming. Automatic evaluation metrics may be used as a proxy during the development of fuzzy matching methods, similarly to the modus operandi in the development of machine translation (MT) systems. A wide range of metrics have been designed for MT since the advent of statistical MT, such as BLEU (Papineni et al. 2002), NIST (Doddington 2002) and TER (Snover et al. 2006). The latter stands for Translation Error Rate[6] and is an edit distance which does not only take account of substitutions, deletions and insertions but also of shifts of blocks of words (which take little effort but change the word order drastically). The minimal TER rate is 0 (no editing required, i.e. sentences are equal).

[6] Originally called Translation Edit Rate.

There is no predefined maximum, as TER results from dividing the number of edits by the number of words in the reference translation. MT metrics can also be used in the context of fuzzy matching, to compare the translation suggestion with the desired translation (Simard and Fujita 2012). This is especially true for TER, as it is expresses the effort needed to correct a translation suggestion. TER can not only be calculated for a single sentence pair but also for a set of sentences. In the latter case, long sentences requiring much editing effort have a stronger influence on TER than short sentences requiring much effort.

## 3. Methodology

In order to measure the usefulness of syntax-based matching methods, we compare their translation suggestions to the desired translation using TER. Our baseline consists of translation suggestions produced by Levenshtein distance on words, as this is the standard matching method. In order for a syntax-based matching method to provide added value, we expect its TER to be lower than that of the baseline, as a lower TER indicates less editing effort. In this regard, we make a distinction between fuzzy match score ranges. CAT tools typically use a fuzzy matching threshold of 70% (Bloodgood and Strauss 2014), which means matches with a Levenshtein score below 0.7 (exact matches have score 1) are ignored, as their translations are considered not useful enough to be provided as suggestions. Therefore, we would like to know whether syntax-based matching methods can make a difference not so much in the high fuzzy range (70 to 100 %) but in lower ranges, in order to improve the recall of the TM, and ultimately improve translators' efficiency.

Applying syntax-based matching methods on a TM requires preprocessing its source sentences. We use the Stanford parser (Klein and Manning 2003, de Marneffe et al. 2006) to parse English sentences, and derive Prüfer sequences from the parses. We mainly focus on the language pair English-Dutch. We match each sentence in the TM enriched with parses and Prüfer sequences to all other TM sentences using the baseline method (Levenshtein on words) and using syntax-based matching methods, more specifically Levenshtein on Prüfer sequences. While Levenshtein distance commonly involves an identical cost for each type of operation (deletion, insertion, substitution), we apply a more sophisticated weighting scheme to elements in Prüfer sequences: we compare features on nodes to one another and combine them in a weighted manner, according to the formula in Figure 4.

$$COST_{Subst} = \begin{cases} 1 & \text{if Term} \rightarrow \text{Non-Term} \\ \lambda_D \Delta_D + \lambda_S \Delta_S + \lambda_L C_L + \lambda_W C_W & \text{otherwise} \end{cases}$$

where

$$\Delta_D = \begin{cases} 0 & \text{if Dependency relations are equal} \\ 1 & \text{otherwise} \end{cases}$$

$$\Delta_S = \begin{cases} 0 & \text{if Syntactic constituents are equal} \\ 1 & \text{otherwise} \end{cases}$$

$$C_L = 1 - \frac{1}{1+\Delta_{Lev_L}}$$

$$C_W = 1 - \frac{1}{1+\Delta_{Lev_W}}$$

$\Delta_{Lev_L}$ is the Levenshtein distance between the lemmas

$\Delta_{Lev_W}$ is the Levenshtein distance between the words

$$\sum \lambda = 1$$

Figure 4: substitution cost of node pair in Prüfer sequence

The comparison of word forms and lemmas is based on a character-based Levenshtein comparison.[7] If two terminal nodes have a different lemma or word form but these are very short (say, determiner *the* vs. *a*), a substitution costs less than if the lemma or word form is long, as the distance is longer in the latter case. The substitution cost also accounts for deletions and insertions. Some examples:

- Comparison of terminal node *hd|VB|fall|fall* with terminal node *hd|VBZ|fall|falls* leads to cost 0+0+0+lambda(word form)*(1-0.5)). The label *hd* stands for syntactic function “head”, *VBZ* stands for part-of-speech tag “verb, third person singular present”, and the last two features are lemma and word form.
- Deletion of non-terminal node *nsubj|NP||*: each feature value is compared to the empty string.

By increasing the two lambdas for syntactic information, we can emphasize the importance of syntactic structure during matching. By increasing the two lambdas for lexical information, we emphasize lexical similarity between sentences. In case of tree alignment distance, we apply a similar weighting scheme when comparing nodes.

In order to speed up matching and avoid having to compare a query sentence to all source sentences in the TM, we build a suffix array from flat sequences using the SALM toolkit[8] and exploit it in a way similar to Koehn and Senellart (2010b). This speeds up the matching process by a factor of 6.

## 4. Results

We applied the methodology in the previous section to a TM which was provided to the SCATE consortium by one of the companies participating in the project. The TM contains about 2,800 sentences (110,000 words). We parsed the English source sentences using the Stanford parser, and derived Prüfer sequences. We used each source sentence in the linguistically enriched TM as query sentence and looked for the highest-scoring match in the remainder of the TM using the baseline matching method, Levenshtein distance on words. The fuzzy threshold was set to 0.2. Figure 5 shows the distribution of the match scores. All matches below 0.2 are considered useless and get score 0. As expected, low-scoring matches are more frequent than higher-scoring ones.[9] After retrieving the translation of the highest-scoring match of a query sentence from the TM, we calculated its TER with the desired translation (the translation of the query sentence in the TM). We also calculated a single TER for the set of best matches in each fuzzy match score range (see previous section on combining TER rates of multiple sentences). Figure 6 shows the TER rate per range (for instance, the leftmost dot refers to range 0.2-0.3). This figure also complies with expectations: TER diminishes as match scores increase.

[7] Characters which only differ in case get a cost of 0.5.

[8] http://projectile.sv.cmu.edu/research/public/tools/salm/salm.htm

[9] Interestingly enough, though, there are far more matches between 0.2 and 0.3 (their frequency is higher than the maximal frequency shown on the y scale) than matches with score 0. Hence, sentences are less likely to be dissimilar to all other sentences than to be slightly similar to some of them.

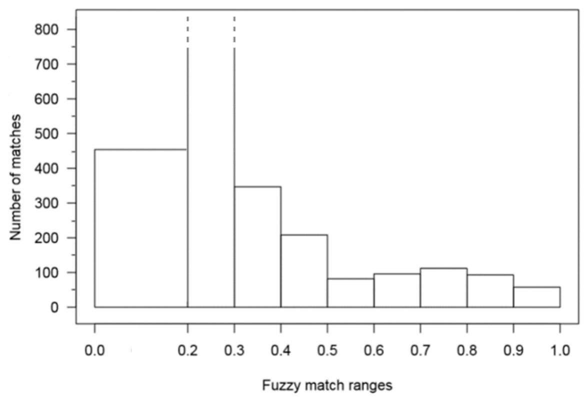

Figure 5: distribution of match scores for total set of best baseline matches in the TM

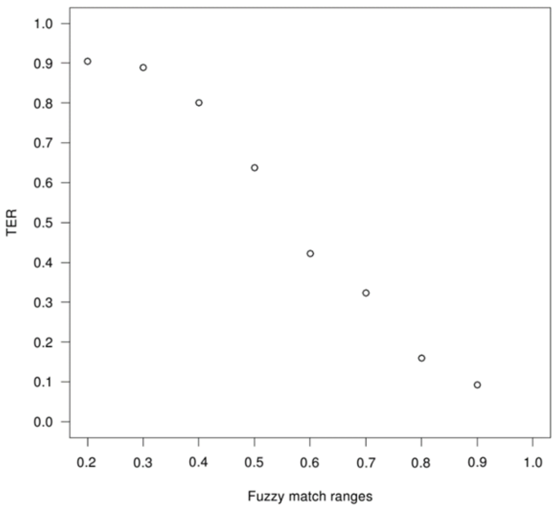

Figure 6: TER for the set of best baseline matches per fuzzy match score range

In order to find out the added value of syntax-based matching for specific fuzzy match score ranges, we applied Levenshtein matching to the Prüfer sequences of the sentences instead of the sentences themselves (we refer to this method as Prüfer matching). We set the lambdas of the weighting scheme described in the previous section higher for lexical features than for syntactic features in order to emphasize lexical similarity. For a given fuzzy match score range, we took all query sentences whose best baseline match had a match score in that range. We then applied Prüfer matching to the same query sentences and calculated TER for the translation of the best matches. Table 2 lists the Prüfer and baseline TER rates per range; rows where rates differ across matching methods are marked in bold.

| range | baseline TER | Prüfer TER |
|---|---|---|
| **0.2 – 0.3** | **0.94** | **0.93** |
| **0.3 – 0.4** | **0.89** | **0.88** |
| **0.4 – 0.5** | **0.80** | **0.79** |
| 0.5 – 0.6 | 0.64 | 0.64 |
| 0.6 – 0.7 | 0.42 | 0.42 |
| **0.7 – 0.8** | **0.33** | **0.34** |
| 0.8 – 0.9 | 0.16 | 0.16 |
| 0.9 – 1 | 0.09 | 0.09 |

Table 2: baseline and Prüfer TER per score range of baseline matches

Although the differences in TER rates in Table 2 are very limited, they indicate that TER tends to be lower for Prüfer matching in lower fuzzy match score ranges. In the upper ranges, both matching methods tend to have the same best matches, leading the TER rates for the range to coincide. Hence, there is a potential for making matches in lower ranges more useful: replacing baseline matches with Prüfer matches in these ranges leads to a decreased effort when correcting translation suggestions.[10] This also holds a potential for improving recall for Levenshtein matching: if lower range matches can be made more useful, one may envisage to lower the minimal fuzzy match score, now typically set at 0.7. However, the small TER rate differences show that there is still a clear need to exploit Prüfer sequences in a better way, for instance by varying more on the weights of lexical and syntactic features and testing on larger translation memories. The tests we are currently performing are still in the exploratory phase.

An example of a query sentence leading to differently ranked match scores for two specific TM sentences is shown in Table 3. The second TM sentence is both syntactically and lexically more similar to the query sentence than the first TM sentence, but the baseline method can only detect part of this similarity because it merely matches word forms. Therefore, it considers the first TM sentence as the best match. Table 4 shows the TER rates of the corresponding translation suggestions; the translation of the second TM sentence is very close to the reference (it only differs in the period).

| Query sentence | Merger falls within scope of Regulation but does raise serious concerns about compatibility with common market | Match score | Rank |
|---|---|---|---|
| TM sentence 1, baseline match | **Merger falls within scope of Regulation but** raises no **serious concerns** | 0.56 | 1 |
| TM sentence 2, baseline match | The merger does fall **within** the **scope of** the **Regulation** and **does raise serious** doubts as to its **compatibility with** the **common market** | 0.48 | 2 |
| TM sentence 1, Prüfer match | Sentence: Merger falls within scope of Regulation but raises no serious concerns<br>Prüfer sequence: … **amod\|JJ\|serious\|serious** dobj\|NP\|\| det\|DT\|no\|no **conj\|VP\|\|** *hd\|VBZ\|raise\|raises* … | 0.81 | 2 |
| TM sentence 2, Prüfer match | Sentence: The merger does fall within the scope of the Regulation and does raise serious doubts as to its compatibility with the common market<br>Prüfer sequence: … **prep\|PP\|\| pobj\|NP\|\| hd\|NN\|market\|market pobj\|NP\|\| amod\|JJ\|common\|common** … **prep\|PP\|\| hd\|IN\|within\|within hd\|VP\|\|** *hd\|VB\|fall\|fall* hd\|VP\|\| aux\|VBZ\|do\|does \|S\|\| **nsubj\|NP\|\|** … | 0.82 | 1 |

Table 3: two TM sentences with different ranking according to matching method

| Query sentence, translation | De fusie valt binnen het toepassingsgebied van de verordening en er bestaat ernstige twijfel over de verenigbaarheid ervan met de gemeenschappelijke markt | TER |
|---|---|---|
| TM sentence 1, translation | **De fusie valt** weliswaar **binnen het toepassingsgebied van de verordening**, maar **er bestaat** geen **ernstige twijfel** | 0.55 |
| TM sentence 2, translation | **De fusie valt binnen het toepassingsgebied van de verordening en er bestaat ernstige twijfel over de verenigbaarheid ervan met de gemeenschappelijke markt** . | 0.05 |

Table 4: editing effort needed to correct the translation of the two TM sentences in Table 3

[10] It should be said, though, that translators using CAT tools may opt for a setting in which they are offered multiple translation suggestions. In such a setting, there should be sufficiently large differences in ranking for a matching method to improve over other ones.

Besides our tests with Prüfer matching, we also performed match tests with tree alignment distance. As the former type of matching is much faster than the latter, the former may act as a filter before tree alignment distance is applied. However, we noted tree alignment distance is prohibitively slow even when only a small amount of parses is involved. Therefore, we need to optimize our implementation of tree alignment distance, which currently performs an exhaustive search for the optimal alignment. A drastic pruning of possible alignment paths will have to take place in order to make application of the method viable, especially when two parses with a large number of nodes are compared. Apart from the problem of speed, we also noted that low parse quality can hamper syntax-based fuzzy matching. Similar sentences may happen to be parsed in very different ways. This problem may be diminished through the use of parse forests, which describe a multitude of parses for a single sentence.

## 5. Conclusions and future research

We have presented a methodology for testing whether the exploitation of translation memories using syntax-based fuzzy matching can improve the recall (by increasing the number of matches) and the usability of TM matches. We compare a sentence to translate, the query sentence, with TM sentences through standard Levenshtein distance on the one hand (the baseline) and through the parse trees of the sentences on the other. The aspect of matching speed is approached through the use of an index (suffix array) and through the derivation of flat sequences from parse trees, Prüfer sequences, which allow to apply string-based methods. Our tests are still in an exploratory phase. First results indicate that replacing baseline matches by matches based on Prüfer sequences slightly improve the TER of translation suggestions for low fuzzy match score ranges. This shows a potential for improving usability of matches and increasing recall of standard Levenshtein matching. We also performed tests with tree alignment distance, which acts directly on trees, but our current implementation of this matching method is still prohibitively slow.

We will pursue tests by applying matching methods using different settings and optimizing them, by making use of additional, larger translation memories in order to increase the likelihood of finding useful fuzzy matches, and by investigating the use of parse forests (Mi et al. 2008) to overcome parsing errors during matching. We will also create an additional type of translation suggestion, which does not consist of the full translation of a sentence in TM, but from the translation of the matching parts only; this will be realized through the use of word alignment from a statistical MT system.

## References

Bille, P. (2005): A survey on tree edit distance and related problems. *Theoretical Computer Science*, *Volume 337, Issues 1–3*, *9 June 2005*. pp. 217–239.

Bloodgood M. and B. Strauss (2014). Translation memory retrieval methods. In *Proceedings of the 14th Conference of EACL*. Gothenburg, Sweden. pp. 202–210.

Cromieres F. and S. Kurohashi (2011). Efficient retrieval of tree translation examples for syntax-based machine translation. In *Proceedings of EMNLP '11*. Association for Computational Linguistics, Stroudsburg, PA, USA. pp. 508–518.

de Marneffe, M.C., B. MacCartney, and C.D. Manning (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*. Genova, Italy.

Doddington, G. (2002). Automatic Evaluation of Machine Translation Quality using N-gram Co-occurrence Statistics. In *Proceedings of the Second Human Language Technologies Conference (HLT)*. Morgan Kaufmann. San Diego, California, USA. pp. 138–145.

Eisele, A. and C. Lavecchia (2011). Using statistical machine translation for computer-aided translation at the European Commission. *Proceedings of the Third Joint EM+/CNGL Workshop "to the User: Research Meets Translators" (JEC '11).* Luxemburg. pp. 3–12.

Gupta, R. and O. Constantin (2014). Incorporating Paraphrasing in Translation Memory Matching and Retrieval. In *Proceedings of the 17th Annual Conference of EAMT.*

Jiang, T., L. Wang, and K. Zhang (1995). Alignment of trees—an alternative to tree edit. *Theoretical Computer Science*, 143. pp. 137–148.

Klein D. and C.D. Manning (2003). Fast Exact Inference with a Factored Model for Natural Language Parsing. In *Advances in Neural Information Processing Systems 15 (NIPS 2002).* Cambridge, MA: MIT Press. pp. 3–10.

Koehn, P. and J. Senellart (2010a). Convergence of Translation Memory and Statistical Machine Translation. In *JEC 2010: Second joint EM+/CNGL Workshop* "Bringing MT to the user: research on integrating MT in the translation industry", *AMTA 2010*, Denver, Colorado, November 4, 2010. pp. 21–31.

Koehn, P. and J. Senellart (2010b). Fast approximate string matching with suffix arrays and A* parsing. In *AMTA 2010*, Denver, Colorado, October 31 – November 4, 2010.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10 (8). pp. 707–710.

Li, G., X. Liu, J. Feng, and L. Zhou (2008). Efficient Similarity Search for Tree-Structured Data. *Scientific and Statistical Database Management Lecture Notes in Computer Science.* Volume 5069, Springer. pp 131–149.

Mi, H., L. Huang, and Q. Liu (2008). Forest-Based Translation. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, USA. pp 192–199.

Papineni, K., S. Roukos, T. Ward, and W.J. Zhu (2002). BLEU: a method for automatic evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the ACL.* Philadelphia, USA. pp. 311–318.

Prüfer, H. (1918). Neuer Beweis eines Satzes über Permutationen. *Archiv der Mathematik und Physik.* 27. pp. 742–744.

Simard, M. and A. Fujita (2012). A Poor Man's Translation Memory Using Machine Translation Evaluation Metrics. In *Proceedings of the 10th Biennial Conference of AMTA.*

Snover, M., B. Dorr, R. Schwartz, L. Micciula, and J. Makhoul (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of AMTA, "Visions for the Future of Machine Translation"*. Cambridge, Massachusetts, USA. pp. 223–231.