

Paraphrasing In Context Of Computer Aided Translation

Turan Rustamli



Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2013

Abstract

We developed a novel paraphrasing service which is provided within a Computer Aided Translation tool. Our service aims to help human translators to fix automatic translations by suggesting paraphrases to any parts of the translation that they specify.

The originality of our paraphrasing approach is that it uses the search graph data, produced during the machine translation process, as the main data source. To test feasibility of our method we carried out two major evaluation stages. During the first stage we designed an original automatic paraphrasing evaluation tool and used it to assess ten different versions of our paraphrasing approach. In order to support our findings we carried out the second stage, during which a group of four annotators judged paraphrasing results produced by our service. The results of the both stages demonstrate that the information available from the search graph is sufficient to produce useful paraphrasing suggestions.

Taking advantage of the interactivity of the Computer Aided Translation process, we also implemented a novel feature that lets the paraphrasing service to provide improved results by utilising feedback from a user.

Acknowledgements

Many thanks to my supervisor Prof. Philipp Koehn for his direction and advice throughout the dissertation. I also wish to express my gratitude to the Government of Azerbaijan Republic for funding my MSc degree through the “State Scholarship Program on Education of Azerbaijani Youth Abroad”.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Turan Rustamli)

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Overview	2
2	Background	3
2.1	Computer Aided Translation	3
2.1.1	Interactive Machine Translation	3
2.1.2	Post-Editing	4
2.1.3	Bilingual concordancer	5
2.1.4	Other assistance ways	5
2.1.5	Summary	6
2.2	Paraphrasing Approaches	6
2.2.1	Data-driven paraphrasing	6
2.2.2	Multiple translations	7
2.2.3	Bilingual parallel corpus	8
2.2.4	Other data-driven paraphrasing approaches	10
2.2.5	Summary	11
2.3	Decoding and search graph	11
2.3.1	A brief introduction to Statistical Machine Translation	11
2.3.2	Decoding	12
2.3.3	Search graph	13
2.3.4	Hypothesis recombination	14
2.3.5	Pruning	14
2.3.6	Summary	15
3	Design and implementation	16
3.1	Analysing requirements	16

3.2	Preparing data	18
3.2.1	Retrieving search graph dump file	18
3.2.2	Preprocessing data for paraphrasing	19
3.3	Project design outline	20
3.3.1	Sending requests from client	21
3.3.2	Request handler	21
3.3.3	Input aligner	22
3.3.4	Graph query execution	22
3.3.5	Filters, score functions and sorters	22
3.4	Implementation details	23
3.4.1	Implementing input aligner	23
3.4.2	Retrieving partial paraphrases from search graph	27
3.4.3	Merging partial paraphrases	29
3.4.4	Implementing functions	31
4	Evaluation	38
4.1	Automatic evaluation	38
4.1.1	Generating test cases	39
4.1.2	Results of automatic evaluation	40
4.1.3	Analysing automatic evaluation results	42
4.2	Manual evaluation	44
4.2.1	Manual evaluation methodology	46
4.2.2	Analysing manual evaluation results	46
5	Interactive Paraphrasing	48
5.1	Handling user feedback	48
5.2	Evaluation	49
5.3	Specified paraphrasing requests	50
6	Conclusion	53
6.1	Summary	53
6.2	Future work	54
	Bibliography	55

Chapter 1

Introduction

1.1 Motivation

Today by taking advantage of achievements in machine translation, we can read and understand text in languages unknown to us. However, high quality translation still needs involvement of human translators. Computer Aided Translation (CAT) aims to support and assist translators, providing them with tools that partially automate and simplify their workflow. Demand for these tools is huge, each year international organisations such as European Union spend billions on translation services.

Over past years researchers introduced several ways to aid translators. The most popular ones are interactive machine translation and post-editing. The first approach provides translation suggestions as user types, while the second approach initiates input area with a machine translation.

During current project we developed and evaluated a novel assistance way that aims to help translators to improve a machine translation, by paraphrasing any part of the sentence that they specify.

Paraphrasing is a process of representing the same idea with different words. Some recent publications suggest that the paraphrasing and the translation tasks are closely related. Moreover, information used to perform one task, can be useful for another (Callison-Burch, 2007).

In this report we will introduce a paraphrasing approach that efficiently reuses outcomes of the machine translation process. Also, we will describe how paraphrasing performance can be improved within an interactive translation environment, by considering user feedback on the results. Finally, we will present a novel automatic evaluation approach that we used to test feasibility and usefulness of our paraphrasing tool.

To confirm the automatic evaluation results we carried out several user studies. We will share outcomes of these experiments and discuss how they correlate with our previous findings.

1.2 Overview

In this section we provide an outline of the report. We will start Chapter 2 by introducing background material that aims to provide required information about the context of our project. We will describe recent advancements in Computer Aided Translation and relate existing assistance tools to our paraphrasing service. Next we will provide references to previous publications on paraphrasing. We will also discuss some basic concepts of machine translation, focusing on the decoding process. The data structure generated during this process is known as search graph and it is used by us as the main source for paraphrasing.

In Chapter 3 we will describe the work that was carried out in scope of current project. Firstly, we will start by discussing the design of our approach and possible ways of integration with existing CAT tools. We will continue by providing implementation details for each component of our initial design. In this chapter we will also focus on challenging problems that were discovered during design and implementation stages, and their solutions.

We will present our evaluation methodology in Chapter 4. We will provide description of an automatic evaluation process which was carried out by us during the project. Furthermore, we will presents evaluation results for ten different versions of our paraphrasing algorithm. We will continue by discussing advantages and disadvantages of the automatic evaluation. Finally, we will provide results of user studies that confirm our previous results.

In Chapter 5 we will present an improved paraphrasing approach that responds to user feedback in order to provide more relevant results. We will also describe a modification that we applied to our automatic evaluation tool in order to support this new feature. We will finish this chapter by introducing another interactive feature, that lets users to make specified paraphrasing requests.

Finally, in Chapter 6 we will conclude our report and suggest interesting directions for future work.

Chapter 2

Background

The main goal of our project was providing a paraphrasing service within a Computer Aided Translation system by reusing the result of the machine translation process, which is used to provide other types of assistance. Before introducing our approach, we would like to provide some background information about assistance types available within CAT systems, existing paraphrasing approaches and some basic concepts of machine translation.

2.1 Computer Aided Translation

Recently, more and more papers discussing various aspects of Computer Aided Translation are published. This increase can be explained by the fact that efficient usage of machine translation advancements to increase productivity of translators is still an open problem. Various assistance tools have been suggested and implemented. Most of these tools are reportedly increasing translation speed, others help to achieve better quality. In this work we present a novel assistance type, which aims to help translators by providing a paraphrasing service. Therefore we want to start by highlighting how our approach could be used in the context of CAT systems together with other assistance types.

2.1.1 Interactive Machine Translation

One way to assist translators is providing auto-completion suggestions while the translation in target language is being composed. As user input in text editor component changes, system suggests possible translation of the next word or phrase. The first

system implementing this approach was *TransType* (Langlais et al., 2000). *TransType* is providing optional sentence-completion suggestions based on machine translation. Considering user acceptance of the suggestions, system can be improved to provide better results (Barrachina et al., 2009). *Caitra* is another CAT system described by Koehn and Haddow (2009), which also provides an interactive mode.

Our paraphrasing approach similarly reuses the search graph to generate paraphrasing options, however in contrast to Interactive Machine Translation systems we assume that the user already has an initial translation that needs to be improved. This initial translation could be originally produced using an interactive assistance with auto-completion or, alternatively, it could be an automatic translation provided for post-editing.

2.1.2 Post-Editing

Popular type of assistance is providing machine translation output as initial translation that could be edited by translator. In this case for each sentence in the document that is being translated, the system provides multiple translation versions. The user can select one of these versions and edit it in order to achieve a high quality translation. Alternatively, the user can ignore suggested options and start typing the translation manually and possibly taking advantage of an aforementioned interactive suggestion tool.

The effect of post-editing on productivity of the translators was analysed by many researchers, including Guerberof (2009). It was previously demonstrated that post-editing can significantly improve performance of translator in some contexts like legal and information technology documents (Federico et al., 2012). However, in other domains like news and media, initial translations produced for post-editing are far from publishable quality. This output is being manually fixed and improved, by replacing erroneous parts with correct words preserving the meaning. This is one of the situations when our paraphrasing service can help user to find better ways to express the ideas in the corrupted parts.

Our current implementation of the paraphrasing tool can be integrated only with a CAT system that provides post-editing assistance. However our approach requires only information generated during machine translation decoding. Considering this, our service can be easily extended to be used with the CAT systems that provide machine translation driven assistance by other means.

2.1.3 Bilingual concordancer

A bilingual concordancer is a feature used by professional translators to explore alternative translations of particular source language phrases. This is achieved by finding occurrences of the phrase in a bilingual corpus and retrieving their corresponding translations. Analysing usage logs of *TransSearch*, one of the most popular bilingual concordancers, for 6 years demonstrate that translators find this feature useful especially for finding the correct sense in case of highly polysemous adverbials and prepositional phrases (Macklovitch et al., 2008). Our tool in contrast, is supposed to work with target language words. Given one possible translation of a phrase, it finds other alternative translations. Similarly, to bilingual concordancer, this might be useful for finding a translation that preserves the meaning of the original phrase in a better way. We expect that starting a search for alternative translations within the same language, will simplify the this task for users with poor understanding of the source language.

2.1.4 Other assistance ways

Providing various translation options for parts of sentence in the source language, can also help in the translation task. As it was shown by Koehn (2010), using this assistance enables translation with acceptable quality even without understanding the source language. The difference of our approach is that instead of providing different translation options for all parts of the original text, we produce paraphrasing suggestions for a user specified parts of translation. Thus similarly to translation options, our tool can be used by someone, who doesn't know source language. For example it may help to understand unclear parts of the text provided for post-editing, by paraphrasing them into representations that make more sense.

Finally, feasibility of our approach could be improved when used together with other CAT features like idioms and terminology detections. Indeed, if idioms or terminology are marked up in user input, during paraphrasing they can be treated in specific ways. In case of terminology paraphrasing lookup might take benefits of domain specific thesauruses. For idioms the paraphrasing algorithm will consider them as atomic units and will not try to paraphrase them partially.

2.1.5 Summary

In this section, we described how our paraphrasing service could be used together with other assistance types within a CAT system. While our implementation requires data generated by a machine translation decoder, this data is not supposed to be generated separately specifically for paraphrasing needs. The data available after running a translation system for an interactive assistance or post-editing could be successfully reused for paraphrasing needs. We also compared our approach with tools like a bilingual concordancer and translation options, highlighting similarities and differences. And finally, we mentioned how paraphrasing could take advantage of idiom and terminology detectors.

2.2 Paraphrasing Approaches

Here we will summarise outcomes of a variety of data-driven paraphrasing techniques previously available in the literature. We want to acknowledge that essential difference of our approach from the ones listed in this section is that while most of previous studies were focused on paraphrasing generation problem, our approach is considering paraphrasing as a search problem. Paraphrasing generation aims to different find ways of extracting paraphrases from various sources and store them for future usage. Paraphrasing search aims to find paraphrases for a specific phrase that is given in a context of a specific sentence that is being translated by a user. Despite this difference ideas from these approaches were used by us for achieving better results and for developing an automatic evaluation approach that was used to assess multiple versions of our paraphrasing algorithm. In this section we will mainly focus on data-driven paraphrasing approaches reviewed in Callison-Burch (2007).

2.2.1 Data-driven paraphrasing

Originally paraphrasing was considered within multiple problems in Natural Language Processing, including question answering, summarisation and natural language generation. As a result historically there were multiple approaches for paraphrasing including techniques that use formal semantic representation and methods that use grammars. More recent approaches are statistically motivated and make use of various data sources to generate paraphrases. Main data sources that were previously used in literature include multiple translations, comparable corpora and monolingual corpora. A

novel technique introduced by Callison-Burch (2007), describes how bilingual parallel corpora could be used for paraphrase generation, this approach is related to the technique we use for paraphrasing. Similarly, the data we exploit is built during a statistical machine translation process and derives from a bilingual parallel corpus.

2.2.2 Multiple translations

The key idea of the techniques that exploit multiple translations as a data source for paraphrase generation is that translators composing different translations of the same text preserve its original meaning, what makes these translations natural sources for paraphrasing (Barzilay and McKeown, 2001). The first experiments with multiple translation of French classical literature were focused on extracting paraphrases by detecting different phrases in similar contexts. So for example, following French sentence, might be translated into English in two different ways as follows:

**Gènèralement, les gens qui savent peu parlent beaucoup,
et les gens qui savent beaucoup parlent peu**

(Jean-Jacques Rousseau)

People who know little speak a lot, and the people who
know a lot speak little.

People who know little are usually great talkers, while men
who know much say little.

Aligning similar parts of both sentences, we can see that “are usually great talkers” and “speak a lot” appear in same context and therefore they probably have same meaning.

Later researchers applied more complex ways to detect paraphrases in multiple translations, this includes using parse trees and considering parts of sentence with similar syntactic role to be paraphrases (Pang et al., 2003).

While our paraphrasing approach does not exploit these ideas, we used similar technique to generate test cases for automatic paraphrasing evaluation. Instead of multiple translations that are rarely available, we used a machine translation to corrupt natural language sentences, and later used difference between original and corrupted sentences as a test case. This technique similarly decides whether parts represent the

same idea by considering the surrounding context. Our evaluation methodology will be discussed in detail in Chapter 4.

2.2.3 Bilingual parallel corpus

Close ties between paraphrasing and translation were studied by Callison-Burch (2007). In contrast to previous studies that use monolingual data as main source for paraphrase detection, this technique exploits parallel corpus that contain text in two languages. Previously, this type of data was considered mostly for solving statistical translation task. It was demonstrated that similar statistical approach could be applied for solving paraphrasing problem as well.

In case of multiple translations aligned equivalent sentences were detected and used as source for paraphrasing. However, this technique deals with sentences paired with their translations and uses an alternative approach. It finds paraphrases using multiple occurrences of same foreign phrase that has different translations as pointers to possible paraphrases. Indeed, if two phrases have same translation in a given number of cases, they are probably encoding same meaning.

The main idea could be illustrated in terms of probabilities. Let's define paraphrase probability as $p(e_2|e_1)$, which designates a probability of that phrase e_2 is suitable paraphrase for given phrase e_1 . This probability is defined similarly to the translation model probability $p(f|e_1)$, which expresses probability of that foreign phrase f has same meaning as given original phrase e_1 . Another required expression is $p(e_2|f)$, the probability of that phrase e_2 is translation of given foreign phrase f . Considering that original phrase e_1 that we are trying to paraphrase may have multiple foreign translations, for the final expression paraphrase probability we will sum over f .

$$\hat{e}_2 = \arg \max_{e_2 \neq e_1} p(e_2|e_1) \quad (2.1)$$

$$\hat{e}_2 = \arg \max_{e_2 \neq e_1} \sum_f p(e_2|f)p(f|e_1) \quad (2.2)$$

Callison-Burch (2007)

In equations (2.1) and (2.2) \hat{e}_2 represents the most probable paraphrase for originally given phrase e_1 . Multiple ways of computing probabilities $p(e_2|f)$ and $p(f|e_1)$ are studied in context of phrase based translation, for example maximum likelihood estimation could be used in the following way:

$$p(f|e_1) = \frac{\text{count}(e_1, f)}{\text{count}(e_1)} \quad (2.3)$$

$$p(e_2|f) = \frac{\text{count}(e_2, f)}{\text{count}(f)} \quad (2.4)$$

Callison-Burch (2007)

Here $\text{count}(\bar{e}, \bar{f})$ stands for number of times phrase \bar{e} is aligned with foreign phrase \bar{f} , and $\text{count}(\bar{u})$ is number of total occurrences of original or foreign phrase \bar{u} .

There are several problems highlighted by the authors of the approach. Designing our paraphrasing approach we came across similar issues. This can be explained by the fact that the data source we use originates from a bilingual parallel corpus.

Firstly, there are issues with words and phrases that have multiple senses. While the main idea for detecting phrases that share the same meaning is that they should have the same translation in the foreign language, it can be argued that foreign words with multiple senses will probably have different translations for each sense in the source language. An example provided by Callison-Burch (2007), demonstrates how French words *banque* (bank in sense of financial institution) and *rive* (bank in sense of riverbank) both can appear in resulting English translation as word *bank*, and therefore could be mistakenly considered paraphrases. This problem was fixed by a refinement that constrains word sense using context during the paraphrasing.

Another problem occurs when translation in contrast to source uses a non-direct reference or hypernyms in some part of the sentence. This reference could be easily understood within the context in which it was used. However, as surrounding context was not originally considered during paraphrase generation it resulted in suggesting wrong paraphrases like “this organisation” for “European Union”. Other example, could be inaccurate paraphrasing of “President Obama” into “President of United States”. Suggested solution for this issue is adding specific constraints to paraphrases for example considering syntactic category, and agreement.

One more addition to the process of ranking paraphrases, that was suggested by authors was using language model probability considering context. Our approach also uses language model probability as one of the features contributing to the rank of the paraphrase, to do so we substitute phrases with their paraphrases in original sentences and evaluate them using language model. This process will be described in details in Chapter 3.

Finally, the quality of paraphrasing is linked to the alignment quality, which is also crucial for translation quality. It was suggested that using of multiple parallel corpora can reduce effect of systematic misalignment.

2.2.4 Other data-driven paraphrasing approaches

Two following techniques are not directly linked to our paraphrasing implementation. However ideas and heuristics from these methods could be used in order to collect alternative training data for our paraphraser, which due to its modular design can support additional auxiliary data sources.

Comparable corpus approach is using texts that discuss same subject as source for paraphrasing (Dolan et al., 2004; Dolan and Brockett, 2005). For example, news articles that describe the same event are mentioning same ideas and concepts using different words. A different example of a comparable corpus could be articles from different encyclopedias defining same concept. Finding paraphrases in this cases is a more challenging task than in case of previous approach we discussed. But in contrast to multiple translation such data source is easier to find. A string distance feature is used to detect similar parts in these texts. If the algorithm spots high concentration of similar words in two sentences from different texts, it considers the sentences to be equivalent.

Researchers also noticed that in case of news articles, first sentences usually tend to summarise the main body of the text. Considering this, they treat the first sentences from all texts separately. The main intuition behind the heuristics is that the first sentences are more likely to carry the same meaning (Dolan and Brockett, 2005). A parallel corpus is built from resulting sentence pairs. Later the paraphrasing task is solved as monolingual translation problem. The problem with this approach is that after pairing sentences even from large sources, the resulting parallel corpus is significantly smaller, this is explained by the fact that detecting comparable sentence pairs is a complex problem, without an efficient solution.

A paraphrasing generation technique that uses a *monolingual corpus* as data source is primarily based on distributional similarities of phrases. Considering availability of this type of data, researchers proposing the approach suggest that given enough data it is possible to detect paraphrases by analysing patterns in frequencies of given groups of words (Lin and Pantel, 2001). This approach is based on Distributional Hypothesis (Harris, 1954), which was originally proposed for detecting synonyms.

2.2.5 Summary

In this section we listed previous paraphrasing efforts and highlighted their key ideas. In contrast with our approach which aims to provide a targeted paraphrase search, these approaches mainly focus on generation repositories of paraphrases. Ideas from multiple translations approach were used by us during automatic evaluation, some of the refinements applied to the bilingual parallel corpora paraphrasing technique were also used by us to improve output of our paraphraser.

2.3 Decoding and search graph

As it was mentioned earlier, the paraphrasing approach presented in this report reuses information generated during machine translation. Particularly, it utilises a data structure known as *search graph*, that is being built during the decoding stage of machine translation. Here we will briefly review main concepts of Statistical Machine Translation, the decoding process and search graph data structure. This section will also motivate usage of this data structure for paraphrase search.

2.3.1 A brief introduction to Statistical Machine Translation

Statistical Machine Translation (SMT) is one of the fundamental approaches of machine translation, which is currently used to achieve state of the art performance in the field. The main idea is using a parallel corpus which contains aligned phrases in two languages to statistically learn how words and phrases in these languages are related to each other. By employing this information, it is possible to produce translation of any given input from one language to another. This could be expressed using the following Bayes' rule equation, which expresses best translation of an input:

$$\arg \max_{target} p(target|source) = \arg \max_{target} p(source|target)p(target) \quad (2.5)$$

Two main components of this equation are $p(source|target)$, known as *translation model* and $p(target)$, known as *language model*. First expresses the likelihood of a *source* (input in original language) being translation of *target* (part of text in foreign language), and second expresses odds of using *target* in foreign language. While bilingual parallel corpus is required to build translation model, monolingual corpus could be used to produce language model.

The leading SMT approach is known as phrase-based machine translation. In this approach group of words (*phrases*) are used together with individual words in order to achieve the most likely translation. These phrases are not linguistically driven, but rather statistically extracted from corpus. The process of associating words and phrases of source language with corresponding words and phrases of foreign language is known as word alignment. Various techniques were introduced to solve this problem, one of the most successful approaches is GIZA++ that is available within Moses, the translation toolkit that we used during this project.

Word alignments are used to build translation model, which could be represented as *phrase table*. This data structure is used to find best translation of a given source sentence. This process is known as decoding and information collected during it is used by us as main source for paraphrasing.

2.3.2 Decoding

Given an input sentence, the number of possible translations grows exponentially (Koehn, 2009). Even for short sentences it is not possible to score all possible ways of translation. Considering such a large search space it is unlikely to find the most probable translation. It was shown that decoding problem, given required machine translation models, is an NP complete (Knight, 1999). A beam-search decoding algorithm is used to find suitable translations. It explores a search space that is reduced by recombination and pruning techniques. While the main goal of the process is producing one, the most suitable, output in target language, it is also possible to produce a given number of candidate translations that are ranked highest by the search algorithm.

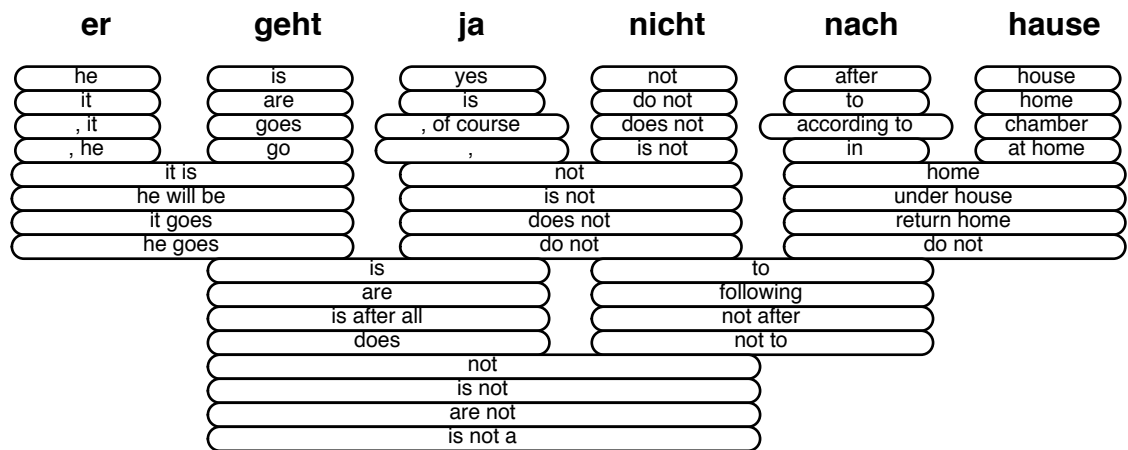


Figure 2.1: Translation Options

Koehn (2009)

2.3.3 Search graph

By grouping words in the input sentence in different ways to form phrases, it is possible to achieve many different translations. Each phrase in source sentence has multiple corresponding translations, which are called *translation options*. Figure 2.1 demonstrates how sample German sentence can be separated into phrases in various ways, and depending on this separation phrases will have different English translation options.

This translation options are used as building blocks in a process known as *hypothesis expansion*. During this process, a data structure known as *search graph* is being built by sequentially joining translation options that cover different parts of sentence in source language (Figure 2.2). Nodes of this graph are called *hypotheses*. As new translation options are being added to existing hypotheses, new partial translation is formed (Koehn, 2009). This process continues until the sequence of hypotheses covers all parts of the input sentence. Each hypothesis contains information about the translation being attached in the current step, the total score of the current partial translation, a link to the previous hypothesis that is extended by the current one and change in the coverage vector of the foreign language sentence.

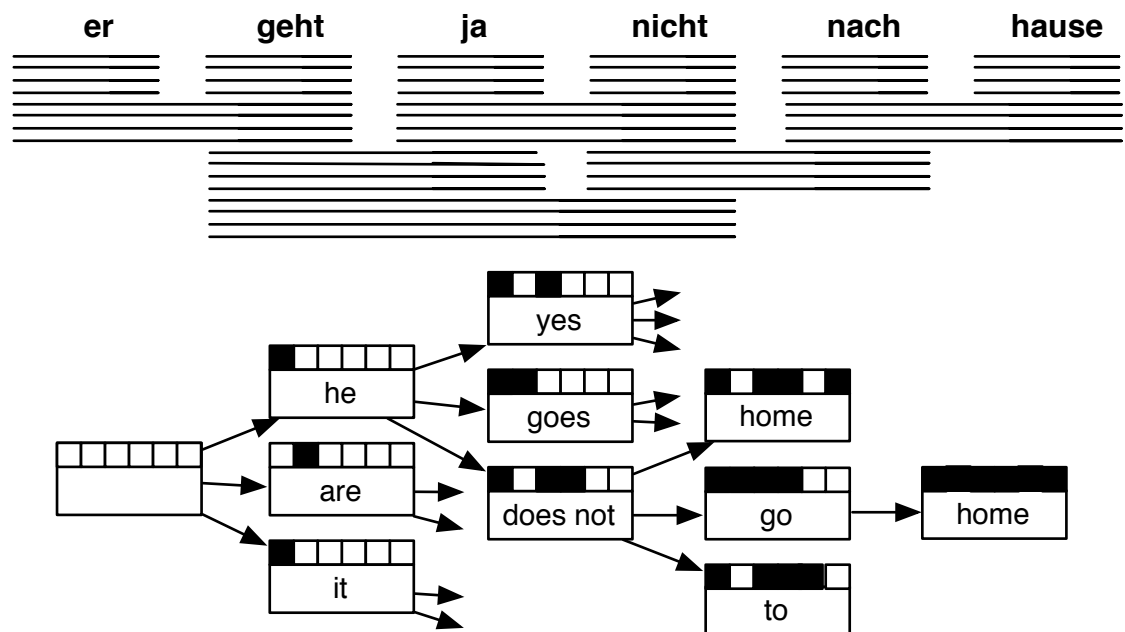


Figure 2.2: Hypothesis expansion

Koehn (2009)

2.3.4 Hypothesis recombination

The resulting data structure is still too large to be directly used for translation search. Several techniques are applied in order to reduce its size. One way is using *hypothesis recombination*, which merges paths with same output and foreign language coverage. This reduction is risk-free and may not cause losing a good translation option. Sample recombination is illustrated on Figures 2.3 and 2.4. Both hypotheses before recombination produce the same output and cover same parts of source sentence, therefore the one with lower score could be recombined to reduce search space. Although this technique removes many unnecessary paths from graph, applying recombination does not affect overall exponential growth in number of hypothesis.

2.3.5 Pruning

Further reduction of the search graph is required for longer sentences. *Stack decoding* is applied in order to filter hypothesis that have low probability of leading to good translations. Essential difference from recombination is that, this reduction technique is not risk-free and potentially may eliminate some useful translations. During this

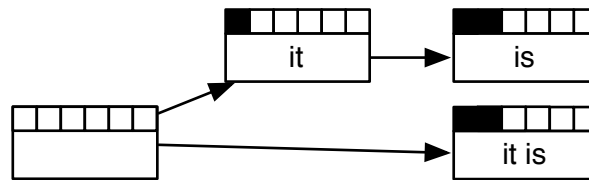


Figure 2.3: Before recombination

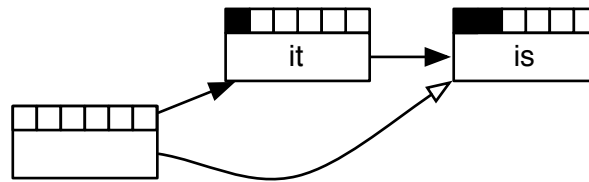
Koehn (2009)

Figure 2.4: After recombination

Koehn (2009)

process stacks containing hypotheses with same coverage are constructed and pruning methods like histogram and threshold pruning are applied. As result number of items in stacks are reduced.

2.3.6 Summary

In this section we provided a brief introduction into phrase-based Statistical Machine Translation, focusing on decoding stage during which search graph is being produced. Two reasons of erroneous translation produced by the process we described are *search error* and *model error*. While the first one corresponds to failure of finding the highest scoring translation during search, the second one refers to initial problems in the machine translation model. The assistance tool developed and investigated by us aims to help users to fix these errors within a Computer Aided Translation system by pointing to erroneous parts and selecting a better paraphrasing option from the suggestions list. In some sense our approach lets user reuse information collected during decoding. This information is available as search graph output, which optionally could be returned by Moses, together with translation results.

Chapter 3

Design and implementation

In previous chapter we discussed how multiple types of assistance provided within Computer Aided Translation systems relate to our novel assistance tool. Further, we reviewed previous paraphrasing approaches highlighting their similarities and difference with technique we use to search paraphrases. And finally we provided a brief overview of phrase-based Statistical Machine Translation to introduce origins of the search graph, the data structure used by us as main source for paraphrasing.

The current chapter will provide detailed description of work that was carried out during the project. We will start by analysing requirements for a paraphrasing service within CAT environment. After that we will describe an iterational design and implementation process we applied to develop working version of such service. Major design decision that were made during this work will be introduced and justified.

3.1 Analysing requirements

As mentioned earlier, an essential aspect of our paraphrasing approach is that it aims to provide a real-time targeted paraphrase search. Unlike previous efforts discussed by us before, we consider an interactive environment where queries are sent by users and executed against our data model. The results of these queries after being dynamically filtered and ranked using various heuristics are returned back to user. In this section we will list main requirements that were considered during the design stage.

Integration with existing CAT software. Taking into account that most of modern CAT implementations are providing web based service (Caitra, CASMACAT, MateCAT), we also employed the client-server architecture to design our paraphrasing service. As a result it might be easily attached as a module to the services provided

online. Moreover, our service could be hosted as a standalone service and integrated with desktop translation software by providing an API. This way our paraphrasing assistance might be accessible from all major types of CAT applications.

Reusing results of machine translation. Another important feature of our paraphrasing system is being able to produce paraphrases using only data available as a result of machine translation carried out to provide other assistance types. We designed our implementation to support translation output generated by Moses (Koehn et al., 2007). The process of retrieving and processing search graph will be reviewed by us in Section 3.2.

Importance of ranking. Quality of final ranking of the paraphrasing options is crucial requirement for our tool that aims to provide an interactive service. Indeed, basic concepts of Human-Computer Interaction suggest that paraphrasing options displayed to users will be efficient only in case if there are no more than seven options on the screen at a time. This means that it is important to rank the top results in a way that they will contain at least one good paraphrase. Features that we exploited for ranking as well as a dynamic system of filters and sorters that we applied to improve top results will be described in Section 3.5.

Paraphrasing granularity. Various levels of paraphrase granularity exist. In case of entire sentences the task is known as *sentential* (or *clausal*) paraphrasing, for shorter items it is called *lexical* (or *phrasal*) paraphrasing. For our project we investigated only lexical paraphrases, motivating this decision by the fact that other assistance tools within a CAT system may already provide multiple translation options for the entire sentence. For example, results of translation by using different bilingual parallel corpora could be provided. We also acknowledge that within scope of lexical paraphrases there is a further separation into two classes that correspond to shorter and longer phrases. Both cases were considered by us during evaluation.

We also took into account the findings presented in Simard and Macklovitch (2005), which analyses usage patterns of another assistance tool, a bilingual concordancer. Authors suggest that average length of the input query is 2-3 words. Parallels between this assistance way and ours were discussed in previous chapter. Considering these similarities we expect of average paraphrasing query length to be the same.

Realtime experience. Another requirement for our interactive tool is providing paraphrasing options as fast as possible. Our tool will not be useful if returning results takes longer than time user spends on manual paraphrasing.

Coexistence with manual editing. While our tool aims to be used in order to

fix erroneous part in the translation by getting the corrected paraphrase automatically, we acknowledge that final high quality translation cannot be achieved only by using paraphrasing. An ability to manually post-edit translation is crucial. Designing our approach we considered that it will be used in an environment where translation could be altered by user at any time.

3.2 Preparing data

3.2.1 Retrieving search graph dump file

A representation of the search graph, described by us in Section 2.3, could be optionally returned by Moses using `-output-search-graph FILE` parameter. The resulting text file contains list of hypotheses that were considered during the decoding stage. Each line in the file represents such hypothesis. Content of file is ordered by sentence identifiers that appear in the beginning of each line. This identifiers indicate to the sentence that was being translated when the hypothesis formed. The hypotheses are in one of three possible formats.

The *initial hypothesis* refers to the empty hypothesis that is used as starting point in process of hypothesis expansion. This hypotheses have a simple structure and don't carry any additional information:

```
0 hyp=0 stack=0
```

The *regular hypothesis* encodes information about translation decisions that were made by the decoder in a given state. Sample regular hypothesis has the following format:

```
0 hyp=10 stack=1 back=0 score=-1.497 transition=-1.497 forward=3437
fscore=-8.900 covered=0-0 out=del Parlamento
```

The line starts with sentence identifier which is followed by a unique hypothesis identifier `hyp=10`. This identifier is used to reference hypotheses. The stack in which hypothesis is placed during stack decoding is identified by `stack=1`, this is also the number of words covered in original input.

The next attribute `back=0` is a reference to the previous hypothesis, in this case it refers to the initial empty hypothesis. This means that current hypothesis represents the first addition of a translation option. Overall score of current partial translation is expressed as `score=-1.497`, which derives from a log-probability, calculated

given machine translation models. The cost of transition to current state is available as `transition=-1.497`.

After finalising the translation, hypothesis attributes are enriched by adding information about the best forward step to the end of the graph `forward=3437` and the best score after this step `fscore=-8.900`. Original sentence coverage information is available in the format `covered=0-0`, where 0-0 indicates covered interval. Finally, the last attribute `out=del Parlamento` contains the translation option that was added by introducing current hypothesis.

The third type of hypotheses are *recombined hypotheses*. These hypotheses are omitted as a result of recombination process that was reviewed by us in Section 2.3.5. Recombined hypotheses have following format

```
0 hyp=734 stack=2 back=24 score=-2.684 transition=-1.226
recombined=731 forward=8037 fscore=-7.962 covered=3-3 out=de Apoyo
```

Here the additional attribute `recombined=731` indicates to the hypothesis with a better score, with which current hypothesis was recombined.

Another option to fetch the information generated during decoding is executing Moses with `-verbose 3` parameter. This way detailed logs will be provided by the system. The resulting output will contain mostly discarded hypotheses and will not have information about the best path that is being added in case of the method we described earlier. That is why we prefer using the `-output-search-graph` option instead.

3.2.2 Preprocessing data for paraphrasing

For initial testing purposes we used Moses to translate a set of 3000 Russian sentences from the news domain (*newstest2012b*) into English. The resulting search graph dump file was about 5.7GB. In order to process it efficiently we decided to encode information from this file into a relational database. We developed a small Python script that parsed each line in the search graph file and mapped the attributes into a dictionary. The dictionary was then stored in an SQLite database. The resulting transformation reduced size of our graph representation to 2.3GB. Moreover, main benefit of having our graph stored in a relational database was being able to run SQL queries against it.

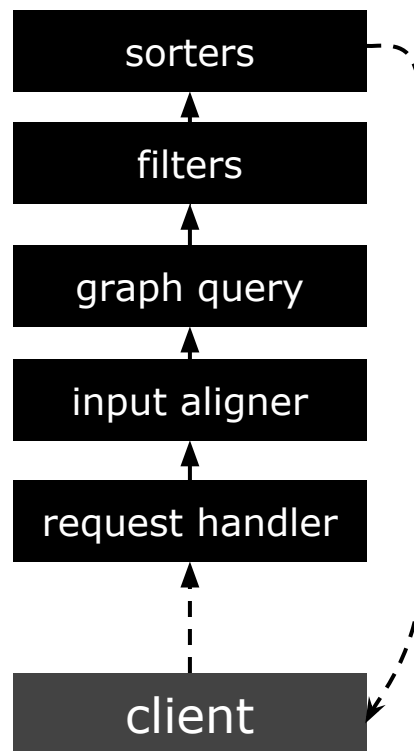


Figure 3.1: Paraphrasing request handling

3.3 Project design outline

In this section we want to introduce high-level structure of the project, listing its main components with short descriptions. The following sections will provide more details about implementation of these components. As it was mentioned before, our service is designed as client-server architecture. The client-side implementation may vary depending on particular CAT software within which paraphrasing service is provided. That is why integrating our solution with a particular CAT system was out of the scope of current work. However we implemented a simple web based interfaces to test usability and feasibility of our tool.

Our server-side implementation is also designed as a proof of the concept and currently supports only paraphrasing of sentences that are available in the preprocessed search graph database. The production of this graph was reviewed in the previous sections. In practice it could be executed as an initialisation step triggered after the machine translation for post-editing is produced. The simple sequence of components that are used to process paraphrasing requests are illustrated in Figure 3.1.

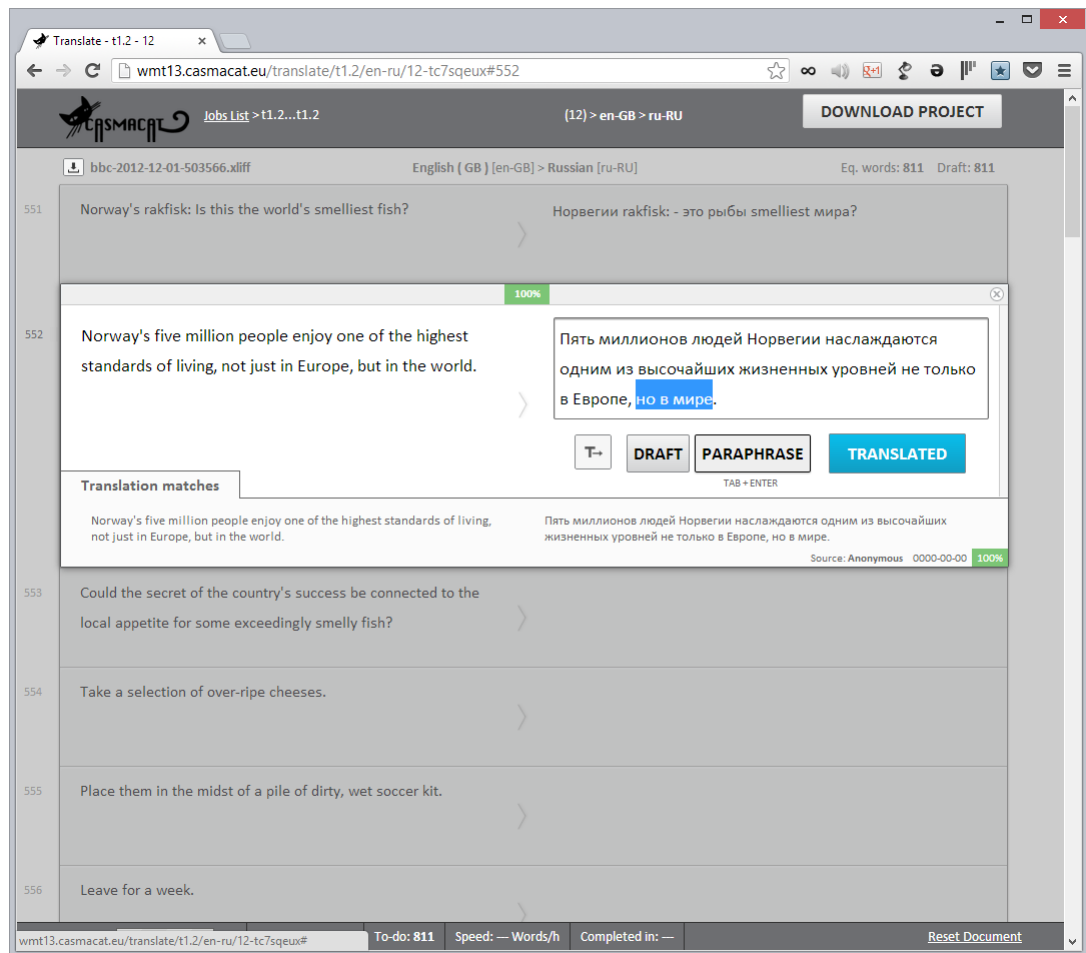


Figure 3.2: Sample screenshot of paraphrasing client UI

3.3.1 Sending requests from client

Simple integration of paraphrasing service into a Computer Aided Translation interface will require adding a button labeled “Paraphrase Selection” near the active translation edition box. The button will be enabled only in case if user selects part of translation text using mouse cursor. Once clicked, button will fire an event that will send a paraphrasing request. The requests from clients are sent over HTTP protocol using AJAX technology. Figure 3.2 illustrates English to Russian translation setup in the CSMACAT system, where user selects part of translation provided for post-editing and paraphrase button is activated.

3.3.2 Request handler

The role of request handler is simply to intercept messages from client-side and to transform them into native easy to use data structures. In our implementation we used

Werkzeug WSGI toolkit which is freely available for Python.

3.3.3 Input aligner

Before the user's paraphrasing query is executed against our graph database, it is being aligned with corresponding part of the original source language sentence. The main reasoning behind this step is making sure that our paraphrasing will not produce duplicated translation by capturing partial translation of a phrase that was not originally requested by user. In this process we are interested only in retrieving pointers to the source sentence parts that correspond to the user selection. Fetching source language phrases is not required.

3.3.4 Graph query execution

Having the required information about the source span the system executes a query against the search graph database. This is achieved by transforming search constraints into an SQL query. Depending on phrase length and other phrase specific features several sequential SQL queries could be executed. The results represent a list of unfiltered translation options that have high probability of being suitable paraphrases for the user selection. Each translation option is paired with data about corresponding hypothesis scores, that are used in further stages of the paraphrasing workflow.

3.3.5 Filters, score functions and sorters

The results retrieved from the graph are usually very redundant and have formatting issues. Thus the next step is filtering them. After that sorter functions are used to generate a final ranking. These steps encapsulate the most significant parts of the paraphrasing algorithm. In order to experiment with various filtering and ranking techniques we have implemented a dynamic system of attachable filters and sorters. This way we were able to easily produce multiple variations of our approach and evaluate all of them against one test suite. Importantly, another benefit of this dynamic modular system is that, depending on user query, we can execute different sets of filters and sorters. For example, some filters could be efficiently used only for short phrases, while others are suitable for long inputs.

3.4 Implementation details

In this section implementation details of the design components will be provided. We will start by consistently discussing the parts that are directly involved in paraphrasing. Description of each component will also contain problematic cases and their solutions discovered by us.

3.4.1 Implementing input aligner

An input aligner detects the coverage of the user selection submitted for paraphrasing. Later, the paraphraser uses this coverage information to ensure that results do not contain words with meanings that are not part of the original selection. Information about coverage is available within machine translation output. The translation output for each sentence is formatted as in the following sample:

```
Indiana |0-0| became the |1-1| first state |2-3|  
to impose |4-4| such a requirement . |5-7|
```

Here, each phrase in translation is followed by coverage details. These details are surrounded by vertical bars and contain start and end positions separated by a hyphen. For example, from this part: `first state |2-3|`, we can see that “first state” phrase corresponds to words in the original sentence with indices 2 and 3. We are only interested in these indices as the corresponding words do not carry useful information for paraphrasing.

The alignment is achieved by matching the user’s paraphrasing input with the machine translation output sentence. For now, we consider that user’s selection matches part of unmodified translation output. Later we will describe cases when user requests paraphrasing after manually editing the translation. The alignment procedure includes following steps:

1. **Parsing the output sentence.** We start by extracting coverage information from the output sentence and storing it in a separate data structure. The coverage data structure represents mapping between translation output and original sentence expressed in word indices. It is also possible to execute this transformation during data preprocessing described in previous section.

2. **Locating input.** On this step we match user's paraphrasing input with the translation. We express input words using indices of corresponding words in translation.
3. **Finding involved intervals.** The final step is merging results of previous two steps to find indices of words in original sentence that correspond to user's selection.

For example, application of step 1 to our sample sentence will produce the following coverage mapping:

```
|0-0| : |0-0|, |1-2| : |1-1|, |3-4| : |2-3|,
|5-6| : |4-4|, |7-10| : |5-7|
```

On step 2, given became the first state as user's input, we will get:

```
became [1], the [2], first[3], state[4]
```

Considering that user can submit only one selection at a time this mapping will always be sequential. Finding corresponding interval for each index in the input representation, we get following coverage information:

```
became the first state |1-3| (1-1, 2-3)
```

Having this information we can consider the alignment to be complete. The described process is straightforward in case of our example. However there are situations when additional heuristics are required for paraphrasing alignment.

One minor problem during alignment is dealing with *selections that contain only some parts of words*. An example of such selection could be `rst state`. This situations are prevented on client-side by appending the rest of the word in case if most of it is selected and omitting the partial otherwise.

Another problem is with *selections that contain only some parts of phrases*. Coverage information available for translation is provided on phrase level and do not contain word by word mapping. For user's input `became the first`, our aligner will produce the same coverage information as for `became the first state`. However, the word with index 3 in original sentence means "state" and it is not covered by `became the first`. During current project this problem was solved by automatically extending user input to match the whole phrases. We also considered one exceptional case, when

the partial phrase contains only function words. Given the first state as paraphrasing input, instead of extending it into became the first state, we reduce it by removing the. Alternatively, this problem could be tackled by using additional detailed word alignment information.

Finally, alignment logic gets more complicated in the case when the user submits a selection after manually editing the translation. In this case, the user input may not exactly match any part of translation output. If user changes are minor, for example fixing agreement between words, our alignment approach can still be used by introducing *fuzzy matching*. These matching is based on edit distance. While this practice is dangerous in case of short selections, with multi-word inputs we can take into account the fact that words should match parts of the translation output sequentially. For example, given told us that war as input, matching the word us should be correct for the following sentence:

the us president told us the war on terror must come to an end

The logic of such fuzzy alignment can be expressed in terms of Hidden Markov Models (HMM). As these models were not used in the core paraphrasing implementation, we will not provide detailed definition of them. In short, HMMs are used to model systems that represent a process with hidden states, while for these states it is true that future and past states are conditionally independent given a number of present states.

For the alignment case, each word in the translation output represents an observation. We denote observations as w_i . Each observation corresponds to one of the hidden states that are words in user input, plus the special state “none”, which means that observation does not correspond to any input word. The states are denoted as s_j .

The states sequence s_1^n with highest probability given observations sequence w_1^n can be expressed using the Bayes’ rule in this way

$$\arg \max_{s_1^n} P(s_1^n | w_1^n) = \arg \max_{s_1^n} \frac{P(w_1^n | s_1^n) P(s_1^n)}{P(w_1^n)} \quad (3.1)$$

Considering a first order HMM, we have following Markov assumption:

$$P(s_i | s_1^{i-1}) = P(s_i | s_{i-1}) \quad (3.2)$$

Having this assumption we can represent prior probability $P(s_i^n)$ as

$$P(s_i^n) = \prod_{i=1}^n P(s_i | s_{i-1}) \quad (3.3)$$

Finally considering the following independence $P(w_1^n | s_1^n) = \prod_{i=1}^n P(w_i | s_i)$; the fact that $P(w_1^n)$ is invariant and prior probability representation from previous equation, we get this expression:

$$\arg \max_{s_1^n} P(s_1^n | w_1^n) \approx \arg \max_{s_1^n} \prod_{i=1}^n P(w_i | s_i) P(s_i | s_{i-1}) \quad (3.4)$$

Probability $P(w_i | s_i)$ from the last equation is known as *sensor model* and probability $P(s_i | s_{i-1})$ is known as *transition model*.

In our alignment case the sensor model is calculated using string edit distance between an observation and the states. The transition model calculation is based on the sequence of words in user input. For our example, the probabilities of the first w_2 and second w_5 occurrences of `us` are calculated as:

$$P(us | w_2) = P(w_2 | us) P(us | none) \quad (3.5)$$

Here we use $P(us | none)$, because we assume that the first word “the” was not matched with any of the input words and thus it was tagged as “none”.

$$P(us | w_5) = P(w_5 | us) P(us | told) \quad (3.6)$$

Here we use $P(us | told)$, because we assume that the fourth word “told” was tagged to be in state *told*.

Here $P(w_i | us)$ expresses probability of that w_i corresponds to `us` from selection. And $P(us | s_j)$ is probability that current word corresponds to `us`, given that previous word corresponds to s_j . Considering the input sequence `told us that war`, the second probability is higher.

In cases when user edits are significant, the described approach will tag all words in translation output as “none”. In these situations, one option is to use a word aligner

like Giza++ to re-align user's current translation with original sentence and use updated coverage information. Another option is to ignore coverage and to switch to a monolingual paraphrasing, this type of paraphrasing is not supported by our approach. Our paraphrasing approach requires the user input to be a partial translation of original sentence. This requirement lets us to use search graph as data source for paraphrasing. Unfortunately, this constraint also does not let our paraphrasing approach to be used outside translation task.

Source code of both regular and fuzzy alignment implementations are available in project file named `aligner.py`.

3.4.2 Retrieving partial paraphrases from search graph

We use the coverage information collected during alignment to construct and execute queries against the search graph database. We formed this SQLite database during preprocessing step. The database contains only one table which has following fields: `sentenceId`, `hyp`, `stack`, `back`, `score`, `transition`, `recombined`, `forward`, `fscore`, `covered`, `out`. These fields correspond to the attributes available from the search graph dump file, that we described in Section 3.2.

Queries encode a number of constraints that are required to find hypotheses that are potential partial paraphrases for user input. These constraints are:

- *Hypotheses should be from the same sentence.* This basic limitation significantly reduces the search space by focusing only on the search graph representation of the sentence that user currently translates.
- *Hypotheses should be within the same coverage as user's input.* As discussed before, it is crucial to preserve same coverage after translation. This rule is related to the paraphrasing approach using bilingual parallel corpora Callison-Burch (2007). Similarly, we consider different phrases that have same translation to be paraphrases. In our case, instead of using parallel corpus, we use information generated for only one sentence. This information encoded in search graph was generated using machine translation models, that were built using parallel corpus. This fact relates our approach to the approach described by Callison-Burch (2007).
- *Hypotheses should be ordered by score in descending order.* This requirement ensures that hypotheses that have better partial score will appear on top of the

list. While having this ordering is useful for filtering and ranking, the score value depends on translation order and that is why it cannot be considered as a sole feature for final ranking decision.

While expressing the first and the last constraints in SQL is trivial, requesting hypotheses within the same coverage requires further analysis.

In order to find all hypotheses that could be used to form final paraphrasing options we have to build queries for each possible separation of the original coverage. However the number of possible ways to split coverage into phrases exponentially grows with number of words covered. For n sequential words in original sentence, there are 2^{n-1} possible ways to group them into phrases. This could be demonstrated by defining split (0) and join (1) decisions between words, then grouping of original words could be expressed as binary sequence with length $n - 1$, where n is number of words. Moreover, the original coverage is not always sequential, therefore we should consider each sequential interval separately.

Considering this exponential growth, starting from cases with $n > 4$ it is not feasible to build SQL queries that reflect each possible grouping. One way to solve this problem is using grouping that was used for producing translation output. We can also use information about the best next step (`forward` and `fscore`) to detect the most useful grouping options. Here are descriptions of the operations that are carried out to retrieve hypotheses that are later used for paraphrasing following input: `first state |2-3|` to impose `|4-4|` such a requirement. `|5-7|`.

- We start by finding *partial paraphrases* for the first phrase in the user selection. In our case this phrase is `first state`. Using SQL we query all hypotheses within current sentence that have coverage `|2-3|`.
- Next, for each result we follow reference to the next best hypothesis and record its coverage. Our goal is to collect all possible coverage types that are available for the best next steps. This way we reuse grouping that was statistically built during machine translation. For our example, two new coverage types are: `|4-4|` and `|4-5|`.
- After removing coverage types that are not within overall coverage of user selection, which in our case is `|2-7|`. We run the procedure described in the previous step recursively for each new coverage type. The results of all queries are recorded in a dictionary data structure, where key represents coverage type

first state |2-3| to impose |4-4| such a requirement. |5-7|

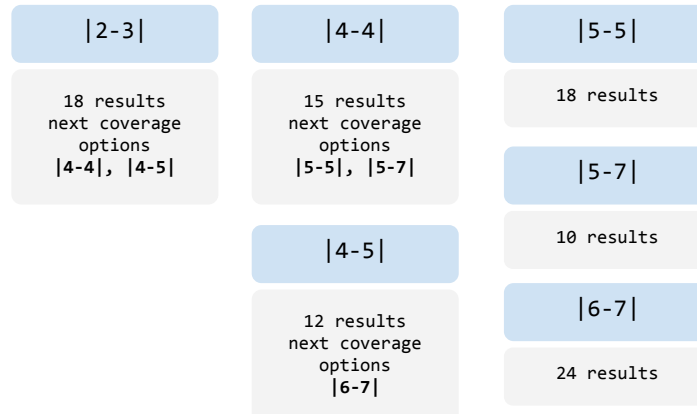


Figure 3.3: Retrieving partial paraphrases with different coverage

and value contains pointers to the rows retrieved from database. For our example in the end we get three new coverage types: |5-5|, |5-7| and |6-7|.

This process is also illustrated on Figure 3.3. While growth of options number in this case is significantly slower than considering all possible groupings, for longer inputs number of queries executed is still too high. Therefore for long selections our approach uses only grouping produced by decoder for translation output. Another option could be splitting longer inputs into small equal parts and then considering each part as a separate paraphrasing input. But this method produces too many partial paraphrasing options and it is not possible to assess all potential ways to join them in context of a real-time interactive system.

3.4.3 Merging partial paraphrases

After running queries we get partial paraphrases stored in a dictionary, where keys represent coverage. The next step is merging this partials in paraphrasing options. We achieve this goal by using a data structure known as finite state machine (FSM). This data structure is defined by set of states, including start/end states and by list of transitions between these states. Our approach is similar to the usage of FSM for finding N-best candidate translations in a search graph which is described in Koehn (2009) Section 9.1.2. In our case we are dealing with significantly smaller part of the search graph which represents the part of the sentence that matches user input, instead of the whole sentence. Also we already have the scores calculated during translation.

The key difference of our approach is that we know that the part of the translation submitted by user for paraphrasing is probably not good enough and we try to reassess scores based on this information and find alternative promising paths in the graph using various features of the hypotheses.

We define one start state which refers to an empty string. Other states represent partial paraphrases (hypotheses) retrieved from search graph. Transitions between states mean adding partial paraphrase to form the final paraphrase. Transitions are only possible in a way that they maintain original coverage. For example, transition from state with coverage $|4-4|$ is only possible to states with coverage $|5-5|$ and $|5-7|$. Additional score is associated with each translation. For each state the score of outgoing transitions are high if they are expected to lead to a good paraphrase, otherwise the score is low. These scores are calculated using various features like string distance, diversity factor and language model score. For each feature there is a *score function* that assesses it for a given transition and adds the result to the final score. The score functions are dynamically attachable, therefore we can easily create many versions of paraphraser that use different combinations of features for assessing transitions.

Before constructing finite state machine, we need to clean up partial results, that contain many redundant results. We use dynamically attachable *partial filters* to filter states considering similarity, function words and punctuation. The next step is adding transitions from start state to states with coverage that matches the beginning of the original overall coverage. We continue by adding transitions considering the overall coverage. The end states of our FSM are the states with coverage that matches the end of the overall coverage. A part of a sample finite state machine for our example input first state $|2-3|$ to impose $|4-4|$ such a requirement. $|5-7|$ is illustrated on Figure 3.4.

After constructing a FSM, the next step is to generate the most promising paraphrasing options. In order to do so, we search for K *best paths* that start in the start state and end in one of the end states. A number of algorithms exist to solve this problem. One of them is *Yen's algorithm* (Yen, 1971). We used its open source Python implementation called *YanKSP*. For each end state we find n best paths, and if we define number of end states as t , as result we get $K = nt$ paraphrasing options. The number of end states t is significantly larger for longer phrases, we compensate it by reducing n . As a result the number of final results K is not too large to be analysed within a real-time system.

Finally, having K paraphrasing candidates our goal is to filter and sort them in

first state |2-3| to impose |4-4| such a requirement. |5-7|

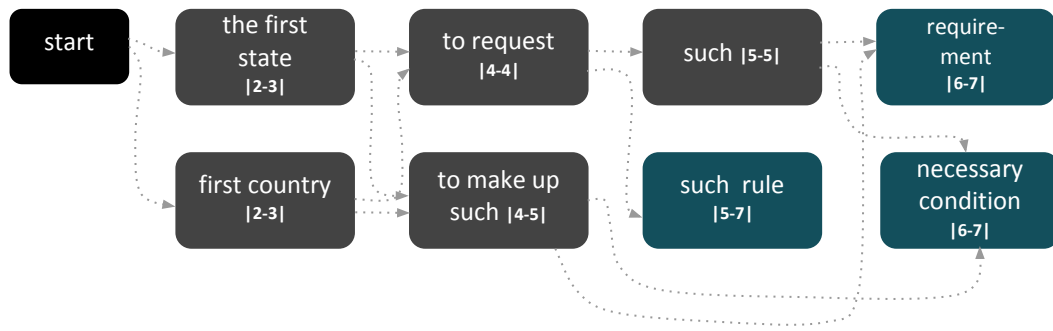


Figure 3.4: A part of a sample finite state machine

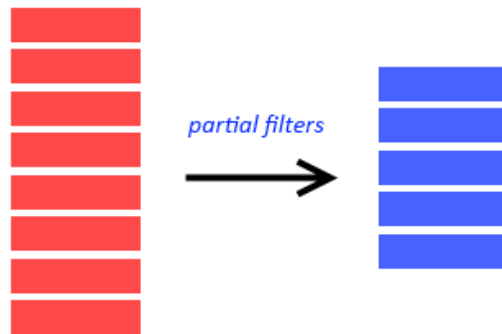
order to achieve the best possible final ranking. Final ranking is crucial because only limited number of top results are displayed to user. The number of results should be small enough to keep paraphrasing decision simple for the user. On this stage filtering and sorting functionality is implemented similarly to partial filters and score functions. Again we have an optionally attachable *filters* and *sorters* that form final list of paraphrasing options. The filters and the sorters, in contrast with the partial filters and the score functions are applied to whole paraphrase candidates, rather than to partial paraphrases, thus they can use more assessment types. At the same time using score functions within a finite state machine has benefits, for example it helps us to preserve calculations carried out to decide whether it worth to merge two partial paraphrases or not.

Figure 3.5 illustrates the workflow described in this section. We will continue by discussing existing filters, score functions, partial filters and sorters in the next section.

3.4.4 Implementing functions

One of the requirements for this project was carrying out a number of experiments to find the set of features within a search graph that are the most useful ones for achieving feasible paraphrasing results. In order to organise these experiments we designed our paraphraser in a way that some parts of its logic are represented by dynamically attachable functions. These functions are grouped by their use cases. Each function within such a group accepts same arguments and returns result in the same format.

1. using partial filters to get the most promising items from partial paraphrases retrieved from the search graph



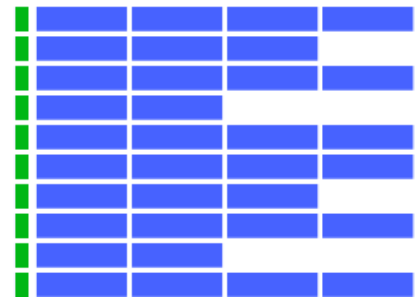
2. the finite state machine built from partial paraphrases transitions are calculated using *score functions*



3. finding K best paths using Yen's algorithm



4. creating initial list of final paraphrasing candidates using K best paths



5. filtering redundant items from initial paraphrasing candidates list



6. using sorters to rank the results in order to get the best paraphrases on top of the list

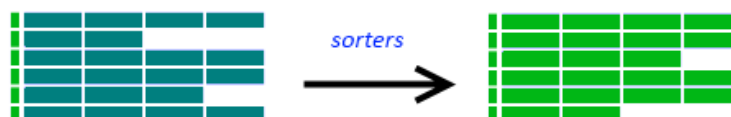


Figure 3.5: This figure illustrates the main workflow of our paraphrasing approach

These groups include: *filters*, *partial filters*, *score functions* and *sorters*. To implement dynamically attachable functionality we used a variation of a classic software design pattern known as *Decorator* (Gamma et al., 1995).

Here we provide a list of the implemented functions. We assign a four letter code for each function. Later these codes will be used to describe functions that were used in different experiments.

3.4.4.1 Partial filters

In previous sections we described how paraphrasing options are built using a finite state machine. States of this machine are hypotheses retrieved from search graph database. Before converting each hypotheses from database query results, we use partial filters to remove the ones that are not likely to be part of a good paraphrasing option. There are two ways to instantiate a partial filter. The first way is to construct it using a list of hypotheses in a format as we get it from database. Items of this list contain all features that are initially available in search graph attributes. After processing this list, partial filter will output a list in the same format with reduced number of items. The second way to instantiate a partial filter is to construct it using another partial filter. In this case before applying filtering logic, our partial filter will execute another partial filter and then will use its results as initial data for filtering. Using both constructors makes it possible to build complex filters. This design solution is the variation of the Decorator pattern, we mentioned before. Following pseudocode illustrates the base class that is extended by all partial filters:

```
abstract class PartialFilter:
    PartialFilter(Hypothesis[] hypotheses)
    PartialFilter(PartialFilter anotherFilter)
    def Hypothesis[] filter()
```

Most of our partial filters aim to reduce original list of hypotheses by removing redundant items. Here we listed partial filters that were implemented during the project:

PTPF: Punctuation Partial Filter

We iterate the input list and remove all punctuation and special signs for each item, also we replace multiple spaces between words with one. We remove duplicates from the resulting list and return the final list.

SDPF: String Distance Partial Filter

For each item in the list we calculate Levenshtein string distance between it and the following items. If the distance is below a given threshold t and it has the same coverage as current item, we remove it from the list. It is important to consider that the scores in the list are already in descending order, so we leave only the best candidates in terms of score. Having same coverage makes these score comparable. For all our experiments we used $t = 3$, because intuitively assumed that filtering items with longer distance we might lose good partial paraphrasing candidates.

FWPF: Function Word Partial Filter

For each item we remove following items if the only difference between two strings is in function words. This way in the resulting list there are still function words. In this filter we use list of functions words available within NLTK library.

3.4.4.2 Score functions

We use score function in order to assess a transition between two steps in paraphrasing finite state machine. Similarly to partial filters score functions follow the Decorator pattern. A basic constructor for a score function requires two hypotheses that refer to source and target states. Score function outputs a normalised value between 0 and 1.

Another constructor instantiates a score function, using other score function. In this case, source and target states are extracted from previous function. Also initial score is set to be equal to the score produced by the previous score function. After calculating score with current function, result is set to be equal to the average of two scores. Pseudocode for base score function class is listed here:

```
abstract class ScoreFunction:
    ScoreFunction(Hypothesis source, Hypothesis target)
    ScoreFunction(ScoreFunction anotherScoreFunction)
    def float calculate()
```

BFSF: Best Forward Score Function

This function is based on similarity with the best next translation option from the search graph. It checks if target hypothesis has the same id as forward reference for the source hypothesis. In this case it outputs 1. Otherwise it retrieves the best next hypothesis and calculates a string distance between it and target. If distance is more

than 9, function outputs 0. Otherwise it the result is calculated as $1 - d/10$, where d is the distance. Current score function uses only the reference to the best next hypothesis (forward) and ignores the value of the `fscore` attribute.

SDSF: Score Difference Score Function

This function deals with the ratio between `transition` scores available from the search graph for source and target states. Output is a number between 0 and 1, calculated as:

$$\min\left(\frac{score_t}{2score_s}, 1\right) \quad (3.7)$$

Here $score_t$ is the target score and $score_s$ is the source score.

This expression encodes an increment in transition score between two states

As a result the output will be higher if transition score for target state in the search graph is higher than the transition score for source state, otherwise it will be lower. The main intuition behind this score function is that if transition score encodes the positive effect of adding particular partial translation during decoding, then following the path along which this score increases should lead to an output that has a good agreement between partial paraphrases.

LMSF: Language Model Score Function

This function uses the language model that was used in the machine translation system to score the result of merging outputs of two hypotheses. In order to get the language model score we use *KenLM* tool in the following way:

```
echo 'this is a test' | ./moses/bin/query ~/lm/bin.lm
```

This command returns a detailed score distribution. However we use only overall total score.

3.4.4.3 Filters

In contrast with partial filters, regular filters are applied to the final list of paraphrasing options. This list is generated using FSM and we still may want to remove paraphrasing candidates that are similar. The interface of this function is very similar to the interface of partial filters. Base class extended by all filters looks like:

```

abstract class Filter:
    Filter(Paraphrase[] options)
    Filter(Filter anotherFilter)
    def Paraphrase[] filter()

```

Two filters implemented by us are repeating logic of the String Distance, Punctuation and the Function Word partial filters. They are correspondingly named **Punctuation String Distance Filter (PTRF)** and **Function Word Filter (FNRF)**. The only novel filter is a score based filter and its description is provided here:

SBRF: Score Based Filter

This simple filter removes items that have a final FSM score which is below a predefined threshold. The filter is used only in cases when number of results is too large to be handled by sorters. Threshold depends on number of items in the list.

3.4.4.4 Sorters

Sorters are applied to the filtered list of paraphrasing candidates. These functions aim to produce best possible final ranking for the paraphrases. They have similar interface with filters, the difference is that instead of removing items, sorters reorder them in the resulting list. Sorters also have parallels with score functions as both function types try to assess quality of paraphrases. While score functions are used to test feasibility of merging two partial paraphrases, sorters are used to assess final whole paraphrasing candidates. Initial list passed to sorters is already ordered by the FSM scores. The parent class for all sorters is listed below:

```

abstract class Sorter:
    Sorter(Paraphrase[] options)
    Sorter(Sorter anotherSorter)
    def Paraphrase[] sort()

```

LMBS: Language Model Based Sorter

In contrast with Language Model Score Function which assesses only partial phrases, Language Model Based Sorter uses the language model score for a whole sentence by replacing user input with a paraphrasing candidate. This way paraphrase is assessed in context of the surrounding words. Items in the list are rearranged by score in descending order.

CDBS: Cluster Diversity Based Sorter

This sorter is based on a heuristic which assumes that more diverse results are more useful for users. Indeed, a user searching for paraphrases is likely to be looking for a way to express idea in a different way, rather than trying to fix a minor spelling mistake. To achieve more contrasting results we use a simple clustering approach. We group items into clusters based on word distance. The distance is calculated similarly to the string distance, considering words as atomic units instead of characters. For clustering we use an implementation of K-Means for Python available within SciPy library. The number of clusters K is set to be equal to the number of top paraphrases displayed to user. The order within each cluster is the same as in the original list. The next step after clustering is to create output list. To do so we iteratively populate an empty list, by adding the top items from each cluster. The order of clusters depends on order of their first item in the original list. This way we achieve an output where top results are significantly different from each other, but at the same time have high scores.

Chapter 4

Evaluation

In this chapter we provide detailed description of the evaluation that we carried out for testing multiple versions of our paraphrasing approach. Considering the modular design of our paraphraser it was possible to design several experiments by adding and removing specific functionality. In scope of this project we were mainly interested in evaluating usefulness of paraphrasing options provided by our tool in context of a translation task. It was not feasible to set up user studies at early stage of the project, because there were too many versions to test. Considering that we designed an automatic evaluation system that helped us to find the most promising versions and also highlighted various issues and interesting features about our tool. We will begin this chapter by describing this automatic evaluation process. At the final stage of project we also carried out a small user study. We will show how results of this study supported our previous findings. Describing each evaluation stage we will also provide analysis and interpretation of the results.

4.1 Automatic evaluation

The design of our automatic evaluation is similar to the multiple translations technique that was described in the background chapter. Our main goal was to test whether our paraphraser can find a suitable paraphrase to a given part of machine translation output. To achieve this goal we created a large repository of test cases. Each test case contains a sentence corrupted by machine translation and a set of suggested paraphrases for different parts of this sentence. In our evaluation process we treat these suggested paraphrases as the gold standard. In order to collect this kind of test data, we artificially corrupted 1000 English (natural language) sentences from news domain

(*newstest2012b* dataset) by translating them into Russian and then back into English. This way the resulting English sentences were expressing the same idea as original ones. However, most of them were stylistically or grammatically incorrect. We detected different parts in original and corrupted version and passed the corrupted parts to our paraphraser. We considered the test case successful if corresponding original part was in top n paraphrasing suggestions. In our experiments we considered $n = 5$, as this is the desired number of options displayed to the user. We begin this section by providing a short overview of the test case generation process, then we will provide the evaluation results. We will conclude current section by discussing problems of this evaluation type.

4.1.1 Generating test cases

In order to generate the test cases, we used the Moses toolkit to translate 1000 English (natural language) sentences into Russian. First, we generated a translation model for English - Russian pair using following command:

```
/moses-git2/scripts/training/filter-model-given-input.pl
~/secondfilter ~/m2config.ini ~/eng.orig.result.in
-Binarizer "/moses-git2/bin/processPhraseTable"
```

Then we used the generated model output (*secondfilter*) to translate sentences executing following command:

```
/moses-git2/bin/moses.2013-01-26 -mbr -mp -search-algorithm 1
-cube-pruning-pop-limit 5000 -s 5000 -threads 24 -ttable-limit 100
-max-trans-opt-per-coverage 100 -f ~/finalfilter/moses.ini
< /fs/vali1/wmt13-en-ru/evaluation/newstest2013.input.lc.1 > ~/rus_out_1
```

Figure 4.1 illustrates sample original English sentence (a) and its Russian translation (b). We used a simple Python script to remove coverage information from the results in order to use it as input for Russian - English translation, sample result is illustrated on Figure 4.1 (c). Following command was used to build Russian to English translation model:

```
/moses-git2/scripts/training/filter-model-given-input.pl
~/secondfilter ~/m2config.ini ~/rus.result.in
-Binarizer "/moses-git2/bin/processPhraseTable"
```

Finally, we translated Russian sentences back to English by running Moses with following configurations:

```
/moses-git2/bin/moses.2013-01-26 -mbr -mp -search-algorithm
1 -cube-pruning-pop-limit 5000 -s 5000 -threads 24 -ttable-limit 100
-max-trans-opt-per-coverage 100 -f ~/secondfilter/moses.ini
< ~/rus.result.in > ~/en_out_new2
```

The resulting English sentence is illustrated on Figure 4.1 (d). As we can see it is different from original English input. In this case the difference is in phrases “us states” and “american states”. Obviously, original phrase “american states” is a better translation and if our paraphraser can suggest it in top 5 results we can consider that as a successful test case.

Considering this, we extracted all differences between original English and corrupted English sentences and stored them in our test case repository. To detect corrupted parts, we used coverage information to align parts of corrupted and original English sentences. After alignment we saved pointers to different parts. For our example, “us states” has coverage $|5-6|$, what corresponds to 6th and 7th words in Russian translation. As we can see from Figure 4.1 (b) these words have following coverages: $|5-5|$ and $|6-6|$. These coverage information points to “american states”. Comparing this to “us states” we spot the difference. We applied the same procedure for all phrases in final English translation. For each sentence we merged sequential difference intervals.

As a result we collected 2139 test cases, that were used for evaluation. While we applied this technique to collect evaluation data, it can also be seen as an alternative paraphrasing approach. Indeed, most of resulting samples were paraphrase pairs. However, this paraphrasing is very expensive and time-consuming to be used within an interactive environment.

4.1.2 Results of automatic evaluation

We tested ten different versions of our approach using the test case repository we discussed in previous section. These versions differ in filters, sorters and score functions. Figure 4.2 illustrates outcome of our automatic evaluation. The first columns identify types of functions we used. Description of these functions could be found in Section 3.4. The final column contains the score, which expresses the number of successfully solved test cases out of 2139 total tests.

- a.** unlike in canada , the american states
are responsible for the organisation of federal
elections in the united states .
- b.** в отличие от |0-1| канады, |2-4| американские |5-5| штаты
|6-6| ответственны за |7-9| организацию |10-12| федеральных
|13-13| выборов |14-14| в соединенных штатах . |15-19|
- c.** в отличие от канады, американские штаты
ответственны за организацию федеральных
выборов в соединенных штатах .
- d.** unlike in |0-2| canada , |3-4| us states |5-6| are responsible for
the |7-8| organization |9-9| of federal |10-10| elections in |11-12|
the united states . |13-15|

Figure 4.1: *Generating test cases.* **a.** The original English sentence from the newstest2012b dataset; **b.** The result of translation into Russian; **c.** The processed Russian translation which is used as an input for Russian to English translation; **d.** The final English translation, which is different from the original English sentence as a result of current process

4.1.3 Analysing automatic evaluation results

As we can see from results illustrated on Figure 4.2, we achieve the best performance with Approach 10. This approach uses all partial filters, all score functions and all sorters in same order as listed in the table. In contrast, Approach 1 has the worst performance. For this approach we did not use any filters or sorters. However we used punctuation partial filter and score difference based score function. We consider Approach 1 as the *baseline* baseline approach, because it uses only minimum required functions. Furthermore, we can see that we achieve better performance by adding language model based features. Also from the results we can see that using sorters significantly improves the performance of paraphraser.

We applied *sign test* to test significance of our results. Considering 0.05 as significance level, we found out that approaches 2 and 3 are not significantly different, as well as approaches 4 and 6. All other approaches were significantly different from each other.

We developed a web based visualisation tool to analyse results of the automatic evaluation in greater details. This tool lists all test cases per experiment. Each item of the list is red in case of failure or green in case of success. We placed “Details” button in each row. Clicking this button launches a modal window which provides more details about the test case, displaying the sentence that was being translated, original paraphrasing selection, desired result and list of suggested paraphrases. This tool was implemented using JavaScript and is currently available on the project website.

Using the visualisation tool we detected multiple problems associated with automatic paraphrases evaluation. Firstly, we noticed that our paraphraser performs significantly better in case of shorter phrases, while for longer phrases it almost always fails to find the desired paraphrase. This dependency is illustrated in Figure 4.3. However, analysing long phrases we noticed that some of them represent good paraphrasing candidates, while they do not match the original natural language English phrases. This results should be considered as success cases for paraphraser, but due to mismatch with original phrase they are classified as failure cases. If a paraphrase doesn’t match the original text, it shouldn’t mean a failure, it still can be a good paraphrasing option. Considering this fact our evaluation scores should be interpreted as *at least N* correct cases out of 2139. In order, to verify results achieved by our automatic evaluation approach we decided to carry out a user study, which will be described in the next section.

Experiments

			Number of correct out of 2139 tests
Approach 1	PTPF	SDSF	135
Approach 2	PTPF FWPF SDPF	SDSF	161
Approach 3	PTPF FWPF SDPF	BFSF	159
Approach 4	PTPF FWPF SDPF	LMSF	211
Approach 5	PTPF FWPF SDPF	SDSF BFSF	161
Approach 6	PTPF FWPF SDPF	SDSF LMSF BFSF	218
Approach 7	PTPF FWPF SDPF	SDSF LMSF BFSF PTRF FNRF LMBS	574
Approach 8	PTPF FWPF SDPF	SDSF LMSF BFSF PTRF FNRF CDBS	493
Approach 9	PTPF FWPF SDPF	SDSF LMSF BFSF SBRF LMBS	661
Approach 10	PTPF FWPF SDPF	SDSF LMSF BFSF SBRF LMBS CDBS	691

Blue - partial filters

Green - score functions

Purple - filters

Magenta - sorters

* Adding Language Model based functions significantly improves results (e.g. App. 3 - 4 and 7 - 8)

* Using sorters for final ranking has the best effect on performance (e.g. App. 6 - 7)

* Using BFSF gives almost the same result as baseline Approach 1, this suggests that this feature probably is not very useful

LMBS: Language Model Based Sorter

CDBS: Cluster Diversity Based Sorter

SBRF: Score Based Filter

FNRF: Function Word Filter

PTRF: Punctuation String Distance Filter

LMSF: Language Model Score Function

SDSF: Score Difference Score Function

BFSF: Best Forward Score Function

FWPF: Function Word Partial Filter

SDPF: String Distance Partial Filter

PTPF: Punctuation Partial Filter

Figure 4.2: Automatic evaluation results

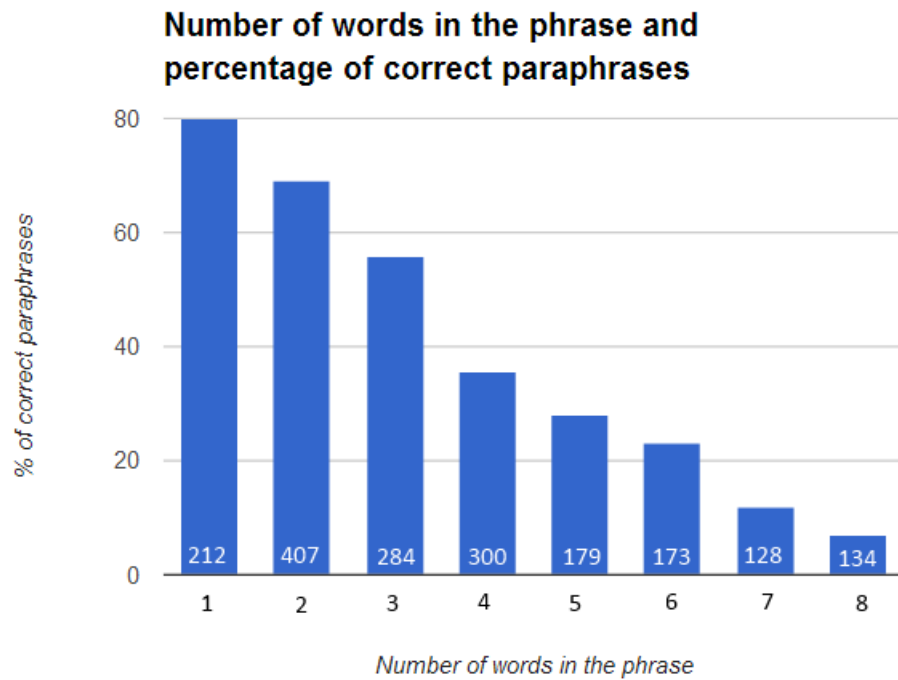


Figure 4.3: Relation between phrase size and paraphrasing performance

White numbers inside bars are the numbers of phrases with corresponding size


4.2 Manual evaluation

During this project we also developed a web based interactive evaluation tool. This tool provides a user with an interface for manual evaluation of paraphrasing results. It reuses automatic evaluation test cases, displaying machine translation of the sentence. The tool also highlights an area that is considered to be corrupted as a result of alignment with original English sentences. Below the sentence we located top 5 selectable paraphrasing options that were returned by a given paraphraser version. User can pick one of these options as a better paraphrase or leave original. We also added an option for user to suggest a better paraphrase if neither options nor original text are suitable translations. Figure 4.4 illustrates the interface of our interactive evaluation tool.

One of our goals was to make this tool easily adaptable for all paraphrasing result data we collected during automatic evaluation. To achieve this we developed a generic interface using JavaScript templating engine Handlebars.js. The data we collected during automatic evaluation stage was stored in JSON-based format and was easily rendered with our JavaScript templates.

project: **rephrase**

[Home](#) [Source](#) [Contact](#)



245

Indiana **became the first state** to impose such a requirement . .

Paraphrasing Options

it was first country

has grown first state

made the first nation

was the first state

led the first country

Original

became the first state

Custom Paraphrase

custom paraphrase

✓ Accept Selection

Submit Custom: Neither original nor options are good

Figure 4.4: *Screenshot of the manual evaluation tool.* This screen displays a single test, where the user has to decide whether paraphrasing options contain a good paraphrase for `became the first state`. For example, the user can select `was the first state` as a suitable paraphrase by clicking on it. This will update the sentence replacing the highlighted area with the user’s selection. In order to submit final decision, the user has to click on “Accept Selection” button

#	A	B	C	D	avg. score
1	8	6	9	6	6/50
7	15	17	12	10	13/50
10	24	20	26	29	26/50

Figure 4.5: *Manual evaluation results*. Here the first column is approach number, the four next columns contain number of successful test cases for each user. And finally the last column is a unified score, which considers a case to be successful if at least two of four annotators decided that the top 5 list contains a suitable paraphrase. We can see that these results correlate with results of the automatic evaluation

4.2.1 Manual evaluation methodology

Considering results of the automatic evaluation, we picked three out of ten tested approaches for the user study. These approaches are 1, 7 and 10. We randomly picked 50 out of our 2139 test cases, ensuring that both short and long phrases present in the final selection. We had four volunteer participants who were provided with a background information about the context of paraphrasing, they were instructed to pick a better paraphrase for selected part of sentence and to press “Accept”, or alternatively to suggest a custom paraphrase and proceed by clicking “Submit Custom”.

All participants were encoded in the following way: A, B, C and D. For each of the three approaches each participant submitted his decisions for the same 50 test cases. We also considered the case when a user comes across a test case that he already assessed for a previous approach and his desired paraphrase selection already is in top 5 for current approaches. We automatically skipped this test cases, considering them successful.

4.2.2 Analysing manual evaluation results

Final results of the user study are provided on Figure 4.5. Collecting decisions from each annotator we calculated a unified score for each approach. This score considers a test case to be successful if at least two of four annotators decided that the top 5 list contains a good paraphrase.

The results of manual evaluation correlate with the automatic evaluation results.

Approach 10 is again the top scoring one. In average annotators decided that it gives a good paraphrase in 52% (26 out of 50) of test cases. For comparison, Approach 7 is judged to be useful only in 26% of test cases. While it is noticeably worse than the best approach it is still better than the baseline Approach 1, which provides a useful paraphrases only for 12% of the test queries. Applying *sign test* with 0.05 as significance level we verified that all three approaches are significantly different and most importantly that Approach 10 is significantly better than the baseline approach.

Chapter 5

Interactive Paraphrasing

Earlier we highlighted that during the current project we considered paraphrasing as an interactive process. We considered interests of the end user by ensuring a real-time performance as well as by focusing on ranking for top paraphrasing candidates. In this chapter we introduce further work in this direction carried out by us during the project. We tweaked our interactive evaluation tool by adding UI elements that could be used to provide quick feedback on results. This feedback is instantly handled by our system in order to produce improved results. We also modified our automatic evaluation system in order to assess usability of this new feature.

5.1 Handling user feedback

Describing paraphrasing process, we introduced a method that uses a finite state machine to output a large number of paraphrasing suggestions. However, on the next steps this list is being filtered, sorted and only top 5 results are displayed to the user. All other paraphrasing options are wasted. Considering an interactive environment, we decided to introduce a UI feature that would let user to redefine his original query in light of the results he receives from the system. Receiving this feedback we go back to the output list and reorder it considering the feedback

Inspired by relevance feedback concept from Information Retrieval field, we added buttons labeled “Show More Like This” near each paraphrase option in output screen. Clicking one of these buttons initiates a server request asking to update results by biasing them towards the corresponding paraphrase candidate. Figure 5.1 illustrates the updated UI.

To implement this feature on the server-side, we simply developed a new sorter

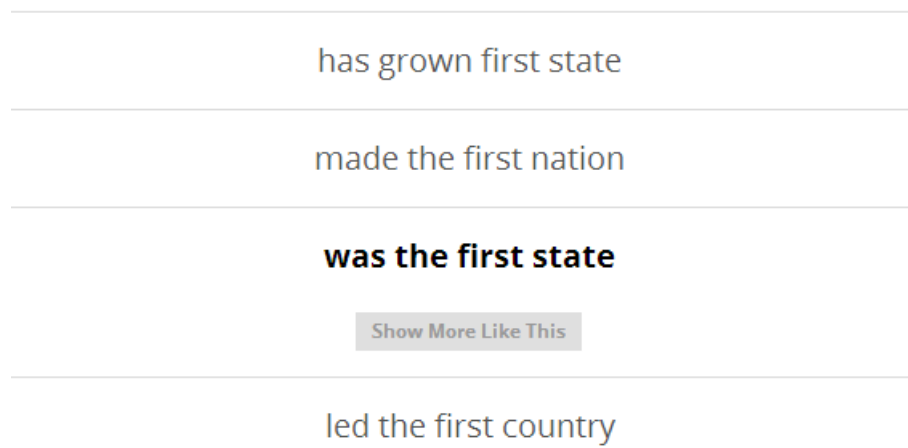


Figure 5.1: “Show More Like This” button

function, named **User Feedback Based Sorter (UFBS)**. This sorter reorders final paraphrasing options list using string distance and word distance features as the primary and secondary weight functions. As a result, top items of the final are the most similar ones to the phrase specified in user feedback.

In current implementation we re-run original query to retrieve the paraphrases list for feedback based sorting. However it is possible to implement this functionality in a more efficient way by saving results in cache, reusing them when user provides feedback.

5.2 Evaluation

We modified our automatic evaluation process, by considering not only the original top results, but also results retrieved by providing each of the original paraphrasing option as user feedback. This way the number of potential paraphrasing candidates dramatically increased. For our baseline approach we got **501** out of 2139 matches with desired paraphrase. Our best performing approach scored **847** out of 2139 test cases. For comparison, without feedback handling feature these numbers are correspondingly **135** out of 2139 for the baseline approach and **691** out of 2139 for the best approach.

However, the automatic evaluation considers that user will always provide useful feedback, which is not a realistic assumption. To test feasibility of the new feature, we carried out another small user study. Two participants were marking 30 same test cases using an updated version of our interactive evaluation tool. For baseline approach we got **11** cases in which at least one user found the desired paraphrase and **6** cases where

both users found a suitable option. For the best approach these numbers are correspondingly **23** and **18**. Sign test confirmed significance of these results. Considering this outcome and comments from the users, we conclude that the user feedback feature is really helpful in context of the paraphrasing task.

5.3 Specified paraphrasing requests

Analysing the situations in which translator may want to use our paraphrasing tool, we found three core use cases:

- The user tries to find a better wording for a translation and wants to explore all options that express the same meaning
- The user expects to get a fixed version of an erroneous part in the translation
- The user is not familiar with the source language and the machine translation does not seem to be good. Paraphrasing is used to understand what the selected part of translation means

Furthermore, investigating each case in detail, we noticed that there are some other special cases when paraphrasing could be useful, including:

- Paraphrasing could be used to expand or collapse abbreviations. As we consider context of translation during paraphrasing this might be useful to find out correct meaning of a given abbreviation.
- Paraphrasing could be used to find out synonyms to a selected words in the translation. This is the main reason behind single word selection requests
- User might want to express same idea in a shorter or longer way, using more or less generic words.

Considering all these situations, we designed another way of adding interactivity to our paraphrasing tool. This is done by introducing a smart “Paraphrase” button, which allows users to specify the main motivation behind their paraphrasing request. The button is implemented using a GUI component known as *split button*, which represents a button with an arrow on the right side. Clicking the button will request default paraphrasing lookup. While clicking the arrow, will suggest more specific options based on the selection. Some use cases of the button are illustrated in Figure 5.2.

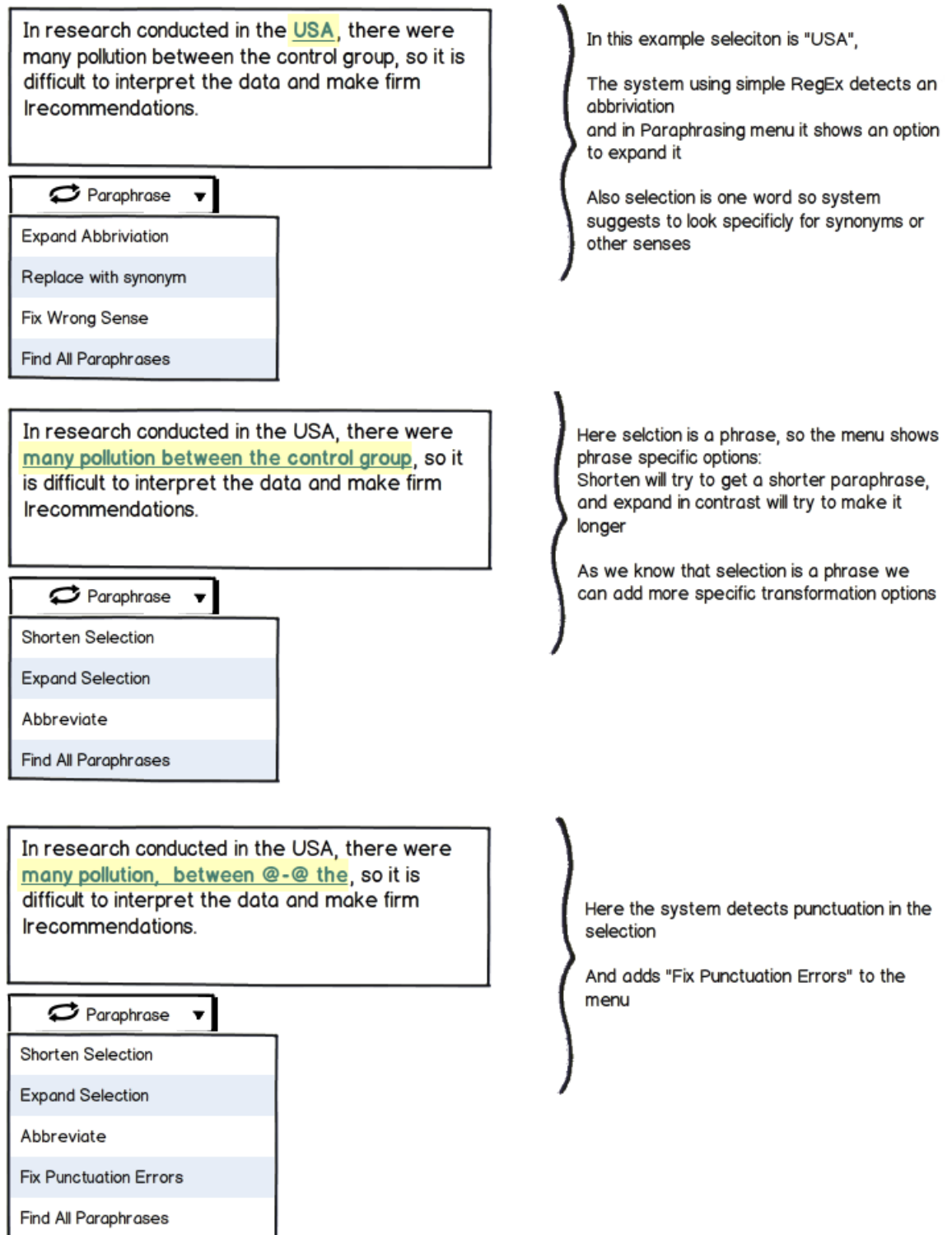


Figure 5.2: "Paraphrase" button

Similarly to user feedback functionality, these kind of specified queries are handled by introducing additional filters and sorters. In scope of this project we implemented only the “expand abbreviation” functionality as proof of the concept. The feature is supported by a filter that removes items where the first letters of each word doesn’t follow the letters in original selection.

Chapter 6

Conclusion

6.1 Summary

During this project we developed and evaluated a paraphrasing tool that could be used as a part of a computer aided translation tool to assist translators. We started by analysing state of the art assistance techniques that are available in recent CAT implementations. Trying to relate these techniques to our approach we noticed that paraphrasing task could be efficiently solved using data produced by other assistance types. We continued our investigation by studying existing efforts in data-driven paraphrasing. We reviewed a paraphrasing approach that uses bilingual parallel corpora as data source (Callison-Burch, 2007). This approach has same high level idea as our solution. The idea is that if multiple items have same translation in foreign language, they are probably paraphrases. In contrast with previous studies, we exploited search graph data as our main source for paraphrasing. This data is being generated during decoding stage of the machine translation process.

We designed a simple paraphrasing service, which aims to find top paraphrases for user input. In Chapter 3 we described the preprocessing step which was carried out by us in order to boost graph querying performance. Next, we discussed importance of coverage information and how user input is being aligned with the foreign sentence. We reviewed the graph querying process, by introducing the coverage splitting problem and a solution we found to resolve it. Finally, our core idea was constructing a finite state machine using partial paraphrases retrieved from search graph and reduced using partial filters. This finite state machine is used to generate best paraphrasing options by using score functions to calculate transition scores between states. Finally, results are cleaned up using a set of filters and final ranking is produced by applying

set of sorters. We implemented partial filters, score functions, filters and sorters as dynamically attachable functions. As a result our core implementation represents a framework that could easily be extended and modified.

To test feasibility of our paraphraser we generated a large test case repository using machine translation corrupted sentences. In Chapter 4 we presented results of this automatic evaluation process for ten different versions of our paraphraser. We also commented on problems with this evaluation approach. Furthermore, we carried out a user study to confirm our previous results. The final outcome demonstrated that our best approach is significantly better than baseline approach which simply produces translation options without applying any sorters or filters.

Finally, in Chapter 5 we introduced additional features that aim to provide better paraphrasing service by taking advantage of the interactive environment. We described our implementation of a feature that allows to instantly update results based on a user feedback. Moreover, we discussed various situations in which paraphrasing tool might be useful and introduced a feature that lets users to request more specific paraphrasing.

In conclusion, we demonstrated how paraphrasing could be efficiently implemented within a computer assisted translation system, by reusing data generated during machine translation.

6.2 Future work

One of the main outcomes of this work is the modular design of paraphraser, which allows to re-implement any part of the paraphrasing process. We previously demonstrated how new features like user feedback and specified requests handling could be easily supported by introducing new filters and sorters. Considering this we believe that our approach can be used as a core for more novel types of paraphrasing. Another direction for future work is collecting paraphrasing usage logs and investigating possibility of using them as training data to improve machine translation. Finally, it might be feasible to extend our approach to support data sources other than search graph, this way it will be possible to consider paraphrasing outside of the translation context.

Bibliography

- Barrachina, S., Bender, O., Casacuberta, F., Civera, J., Cubel, E., Khadivi, S., Lagarda, A., Ney, H., Tomás, J., Vidal, E., and Others (2009). Statistical approaches to computer-assisted translation. *Computational Linguistics*, 35(1):3–28.
- Barzilay, R. and McKeown, K. R. (2001). Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 50–57. Association for Computational Linguistics.
- Callison-Burch, C. (2007). Paraphrasing and Translation Doctor of Philosophy School of Informatics University of Edinburgh.
- Dolan, B., Quirk, C., and Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*, page 350. Association for Computational Linguistics.
- Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proc. of IWP*.
- Federico, M., Cattelan, A., and Trombetti, M. (2012). Measuring user productivity in machine translation enhanced computer assisted translation. In *Tenth Biennial Conference of the Association for Machine Translation in the Americas*.
- Gamma, E., Vlissides, J., Helm, R., and Johnson, R. (1995). Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49:120.
- Guerberof, A. (2009). Productivity and quality in MT post-editing. In *MT Summit XII-Workshop: Beyond Translation Memories: New Tools for Translators MT*.
- Harris, Z. (1954). *Distributional structure*. World.

- Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.
- Koehn, P. (2009). *Statistical Machine Translation*.
- Koehn, P. (2010). Enabling Monolingual Translators : Post-Editing vs . Options. (June):537–545.
- Koehn, P. and Haddow, B. (2009). Interactive Assistance to Human Translators using Statistical Machine Translation Methods.
- Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., and Others (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.
- Langlais, P., Foster, G., and Lapalme, G. (2000). TransType: a computer-aided translation typing system. In *Proceedings of the 2000 NAACL-ANLP Workshop on Embedded machine translation systems-Volume 5*, pages 46–51. Association for Computational Linguistics.
- Lin, D. and Pantel, P. (2001). Discovery of inference rules from text. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 323–328. ACM.
- Macklovitch, E., Lapalme, G., and Gotti, F. (2008). TransSearch : What are translators looking for ?
- Pang, B., Knight, K., and Marcu, D. (2003). Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 102–109. Association for Computational Linguistics.
- Simard, M. and Macklovitch, E. (2005). Studying the Human Translation Process through the TransSearch Log-Files. In *AAAI Spring Symposium: Knowledge Collection from Volunteer Contributors*, pages 70–77.
- Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716.