Paraphrasing In The Context Of Computer Aided Translation

Turan Rustamli

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

Abstract

...

Acknowledgements

Many thanks.

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

Chapter 1

Introduction

1.1 Motivation

Today, taking advantage of advancements in machine translation, we can read and understand text in languages unknown to us. However, producing a high quality translation is a task that is still solved by humans. Computer Aided Translation (CAT) systems aim to support and assist translators, providing them with tools that partially automate and simplify their workflow. Demand for these systems is huge, each year billions are spent on translation by international organisations such as European Union.

Several assistance ways, that could be offered by a CAT system, were introduced since investigation of the question began. The most popular ones are sentence-completion and post-editing. While the first approach is providing suggestions while user is typing translation, the second approach prefills text input area with a machine translation.

In this report we present a novel assistance way that aims to help translator to fix an initial machine generated translation, by providing paraphrases for any user-specified part of the sentence.

Paraphrasing is the process of representing the same idea with different words. Some recent publications suggest that paraphrasing and translation tasks are closely related and information used to perform one task, can be useful for another (ccb). Considering this we are introducing paraphrasing approach that efficiently reuses the information available after machine translation process which output is used to provide post-editing assistance within a CAT system.

Moreover, taking advantage of interactivity of a CAT process, we investigated paraphrasing as an interactive task, implementing a system that accepts and responds to user's feedback on the paraphrases that were initially produced.

We will also describe a novel automatic evaluation approach that we used to test feasibility and usefulness of paraphrasing results. Outcomes of a set of user studies will be provided to back up the results of automatic evaluation.

1.2 Overview

In this section we would like to provide an outline of the report.

We will start the following chapter by introducing background material in the next chapter. Recent advancements in CAT will be presented and references to previous publications about paraphrasing will be provided. We will also discuss some basis concepts of machine translation, particularly Search Graph representation, which will be used by us to solve paraphrasing task.

In the next chapter we will focus on providing description of work that was done in scope of our project. First, we will start by discussing design of our approach and explaining it's possible ways of integration with existing CAT systems. We will continue by providing details of implementation of our design. Description of challenging problems, that were solved during design and implementation stages, and their solutions will be given as well.

We will present our evaluation methodology, stating results of testing various versions of our paraphrasing algorithm in the Chapter 4. This includes description of an automatic evaluation framework which was implemented by us during the project, it's advantages and disadvantages. Next, we will provide results of a set of user studies carried out by us in order to support results of automatic evaluation.

Following chapter will introduce novel experiments by making paraphrasing within a CAT system more interactive. Specifically, we will present a design and implementation of a feature that lets a user to instantly interact with the system and provide improved results. In this chapter we will discuss ”Show More Like This” button that was added by us to the paraphrases list displayed to a user, which is similar to ideas of relevance feedback () in Information Retrieval in order to provide user with more relevant paraphrases. A modification that was made to our automatic evaluation framework in order to support this new feature will be also introduced, as well as results of both automatic and manual evaluation.

Next, we will look into another way of adding more interactivity to the user interface of the paraphrasing functionality, by implementing and adaptive paraphrase button that will let user to initially suggest us the reason of the paraphrasing request. We will discuss situations when users might find paraphrasing functionality useful interacting with a CAT system, providing results of user studies.

In our final chapter we will conclude our report and suggest interesting directions for the future work.

Chapter 2

Background

The main idea of our project is providing a paraphrasing feature within a Computer Aided Translation system by reusing the information that is available as a result of machine translation process which output is used to provide user with an initial translation. Before introducing our approach, we would like to provide some background information about CAT systems, relationship between paraphrasing and translation tasks as well as some concepts of machine translation that were are used by our approach to implement paraphrasing.

2.1 Computer Aided Translation

Recently, more and more papers discussing various aspects of Computer Aided Translation are published. This increase could be explained by the fact that the need of high quality and publishable translation is still satisfied by human translators and how advancements in machine translation are efficiently used by them in order to increase productivity of translators is still an open question. Various ways of assistance were suggested and implemented. Most of these tools are reportedly increasing translation speed, others help to achieve a better quality. In this work we present a novel assistance type, which aims to help translators by providing with paraphrases - alternative ways of expressing the same ideas. Before discussing paraphrasing, we want to highlight the ways our approach could be used in the context of CAT systems that provide different types of assistance.

2.1.1 Interactive Machine Translation

One way to assist translators is providing auto-completion suggestions while he composes the translation in target language. As user's input in text editor component changes, system suggests possible translation of the following word or phrase. The first system implementing this approach was TransType (Langlais et al., 2000; Foster et al., 2002; Bender et al., 2005), providing optional sentence-completion suggestions based on machine translation. Based on user's acceptance of the suggestions, system can be improved to provide better results (Barrachina). Caitra is another CAT system described by Koehn and Haddow (2009) which also provides an interactive mode, where system uses search graph of machine translation decoder to suggest multiple options for upcoming words and phrases translation.

Our paraphrasing approach similarly reuses search graph for generation of alternative translation options, however in contrast to Interactive Machine Translation systems we assume that the user already has an initial translation, that she wants to improve using better wording. This initial translation could be originally produced using an interactive assistance with auto-completion or alternatively it could be a result provided for post-editing.

2.1.2 Post-Editing

Another popular assistance way is providing machine translation for post-editing by translator. In this case for each sentence in the document that is being translated, multiple translation options are provided. User can select one by clicking on it, what will update value of text editor component with the chosen translation. Alternatively, user can ignore suggested options and start typing translation manually and possibly she can take advantage of an aforementioned interactive suggestions.

Effect of post-editing on productivity of the translation process was analysed by many researchers, including (...). It was previously demonstrated that post-editing can significantly improve performance of translator in some contexts like legal and information technology documents (...). However, in other domains like news and media, initial translation produced for post-editing is 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 assistance paraphrasing service aims to help user to find multiple possible alternative translations of the corrupted part.

During this project we were investigating feasibility of our tool, in scope of a CAT system that provides post-editing options. However, our approach requires only information generated during machine translation decoding, the search graph, which is also being build during interactive translation. That means our technique could be easily extended to be used with the CAT systems without post-editing, that provide machine translation driven assistance by other means.

2.1.3 Bilingual concordancer

Bilingual concordancer is a feature used by professional users to explore alternative translations of particular phrases. This is achieved by finding occurrences of the phrase in bilingual corpus and retrieving their corresponding translations. Analyses of usage logs of TransSearch, one of the most popular bilingual concordancers, for 6 years demonstrate that tranlators find this feature useful especially for finding the correct sense in case of highly polysemous adverbials and prepositional phrases Macklovitch et al. (2008). We believe that our paraphrasing tool that given one possible translation of a phrase explorers other alternative translations might be also useful in terms of finding a translation that preserves the meaning of the original phrase in a better way. We expect that starting the search within the same language, will simplify the this task for users with poor understanding of source language.

2.1.4 Other assistance ways

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

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

2.1.5 Summary

In conclusion, we described how assisting translator by providing paraphrases of translation could be used together with other assistance means provided by a CAT system. While our implementation requires data generated by a translation decoder, this data is not supposed to be generated separately specifically for paraphrasing needs, but rather data available after running translation system for an interactive assistance or post-editing could be successfully reused for paraphrasing. We also compared our approach with tools like bilingual concordancer and translation options provider, 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 summarize outcomes of this and variety of other data-driven paraphrasing techniques previously available in the literature. 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, trying to extract paraphrases from various sources and store them for future usage, our approach is considering paraphrasing as search problem and aims to find paraphrases for a phrase that is given in a context of a 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 useful for assessing multiple versions of our paraphraser.

2.2.1 Data-driven paraphrasing

Originally paraphrasing was considered within multiple problems in Natural Language Processing, including question answering, summarization and generation. As a result historically there were multiple approaches for paraphrasing including techniques using formal semantic representation and methods that used grammars. More recent approaches are statistically motivated and make use of various data sources to generate paraphrases. Main data sources that were used in previous literature include multiple translations (for example multiple translations of classical literature), comparable corpora (news articles about same events, different encyclopedia articles with same subject), and monolingual corpora (in this case distributional similarity is exploited to find paraphrases). A novel technique introduced by Callison-Burch (2007), describes how bilingual parallel corpora could be successfully used for paraphrase generation, this approach is related to the technique we use for paraphrasing. Similarly, the data we exploit is build during a statistical machine translation process and derives from a bilingual parallel corpora.

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 translation of the same text were preserving it's original meaning, what makes these translations natural sources for paraphrasing (). 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 savant peu parlent becoup,
et les gens qui savant 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.

While our paraphrasing approach doesn't exploit this ideas. We used similar technique to develop an automatic paraphrasing evaluation framework. 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 paraphrasing target. This technique similarly decides whether parts represent same idea or not by considering the surrounding context. Our evaluation methodology will be discussed in detail in Chapter 5.

2.2.3 Bilingual parallel corpus

Close ties between paraphrasing and translation were studied by Callison-Burch (2007). The results of this work were important for designing our paraphrasing approach.

In contrast to previous studies that use monolingual data as main source for paraphrase detection, technique described by Callison-Burch (2007) exploits parallel corpus that contain text in two languages. Previously, this data has been considered mostly for solving statistical translation task. Authors demonstrated that similar statistical approach could be applied for solving paraphrasing problem as well.

In case of multiple translations and comparable corpora aligned equivalent sentences were detected and used as natural source for paraphrasing. However, this technique that deals with sentences paired with their translations, 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.

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

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

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:

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

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

Callison-Burch (2007)

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

There are several problems highlighted by authors of the approach. Designing our paraphrasing approach we came across similar issues. This could be explained by the fact that data source we use originates from bilingual parallel corpus. We applied variations of the refinements suggested by Callison-Burch (2007) for achieving better paraphrasing results.

Firstly, they introduce issues with words and phrases that have multiple senses. While the main idea for detecting phrases that share same meaning is that they should have same translation in foreign language, it can be argued that foreign words with multiple senses will probably have different translations for each sense in source language. An example provided in Callison-Burch (2007), demonstrates how French words *banque* (bank in sense of financial institution) and *rive* (bank in sense of river-bank) 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 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 that it was used. However as surrounding context is not considered during paraphrase generation it may result suggesting wrong paraphrases like “this organisation” and “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.

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

2.2.4 Other data-driven paraphrasing approaches

Two other techniques described in this section are not directly linked to our paraphrasing implementation. However ideas and heuristics from these methods could be used in order to collect an alternative training data for our paraphraser, which due to it’s modular design can support additional auxiliary data sources.

Comparable corpus approach is using texts that discuss same subject as source for paraphrasing. For example, news articles that describe same event are mentioning same ideas and concepts using different wording. Different example of comparable corpus could be articles from different encyclopedias defining same concept. Finding paraphrases in this cases is more challenging task than in case of previous approach we discussed. But in contrast to multiple translation such data source is easier to find. String distance feature is used to detect similar parts in these texts. So if algorithm spots high concentration of similar wording 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 summarize the main body of the text. Considering this, they treat the first sentences from all texts separately, considering that they probably carry same meaning. A parallel corpus is build from resulting sentence pairs and 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. 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 highlighting their key ideas. These approaches mainly focus on generation repositories of paraphrases, in contrast to our approach which aims to use data not for extraction, but rather for a targeted paraphrase search. Ideas from multiple translations approach were used by us in automatic evaluation framework construction, some of the refinements applied to the bilingual parallel corpora paraphrasing technique were also useful in terms of improving output of our paraphraser.

Moreover, analysing paraphrasing as a interactive search problem, it's possible to reuse some ideas from Information Retrieval field. In Chapter 7 we will introduce a paraphrasing workflow that exploits feedback from users and instantly returns improved results. This feature was implemented using relevance feedback technique that is used by some web search engines to help users to redefine their original queries.

2.3 Decoding and search graph

As it was mentioned earlier, paraphrasing approach presented in this report reuses information generated during machine translation. Particularly, it utilises data structure known as *search graph*, that is being built during the decoding stage of translation. Here we will briefly review main concepts of Statistical Machine Translation, the process of decoding and how it could represented using a search graph. This section will also motivate usage of this data structure for paraphrases 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 currently is 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 and later employing this information 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 in target language of an input in source language:

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

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 knows 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. This 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 knows as word alignment. Various techniques were introduced to solve this problem, one of the most successful approaches is GIZA++ that is available within Moses a translation toolkit that we used during this project.

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

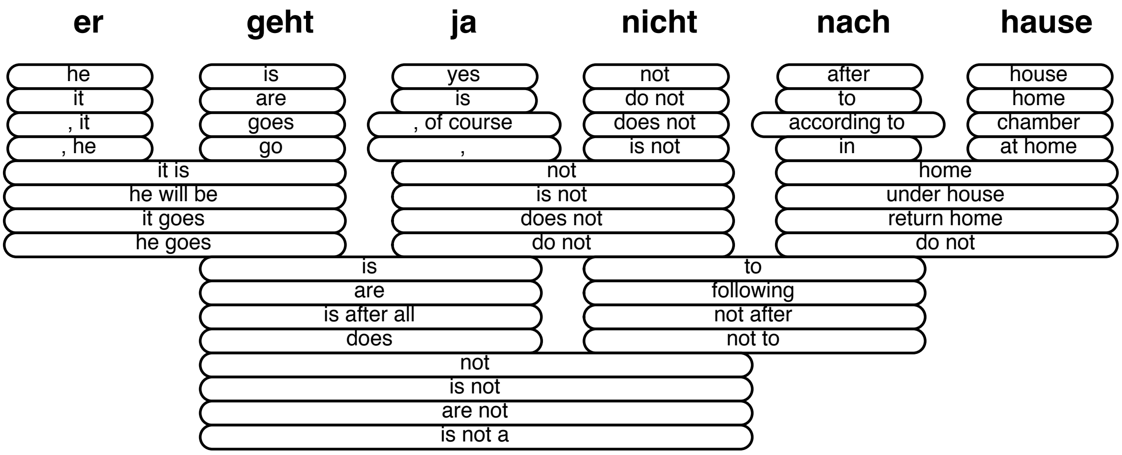


Figure 2.1: Translation Options

Koehn (2009)

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's unlikely to find the most probable translation. It was shown that decoding problem, given required machine translation models is an NP complete [[Knight (1999)]]. However, using various heuristics it is possible to find translation that is good enough for understanding the meaning of the sentence in source language. This is achieved by applying beam-search decoding algorithm that explorers a search space that is reduced by recombination and pruning techniques. While main goal of the process is producing one the most suitable output in target language, it is also possible to produce given number of candidate translations that are considered to be good enough by the search algorithm.

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 phrase in various ways, and depending on this separation phrases will have different English translation options.

This translation options are used as building blocks in process known as *hypoth-*

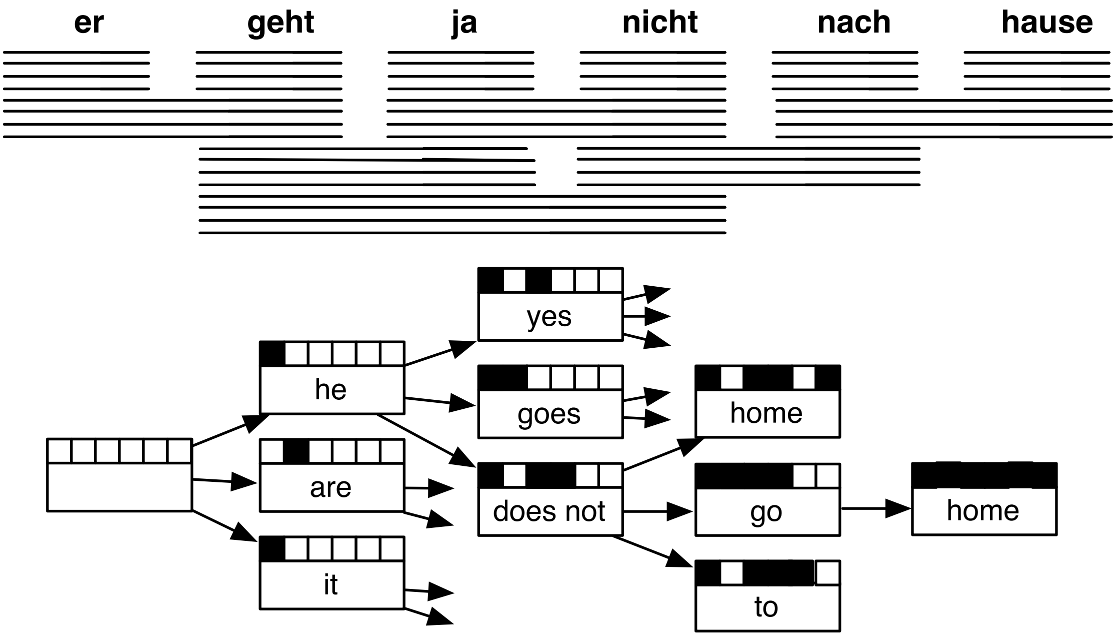


Figure 2.2: Hypothesis expansion

Koehn (2009)

esis 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 sequence of hypotheses cover all parts of input sentence. Each hypothesis contains information about the translation being attached in current step, total score (based on log-probability) of current partial translation, link to previous hypothesis that is extended by current one and change in coverage vector of the foreign language sentence.

2.3.4 Hypothesis recombination

The resulting data structure is still to large to be directly used for translation search. Several techniques are applied in order to reduce it's 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 of recombination is illustrated in Figures 2.3 and 2.4. Both hypotheses before recombination produce same output and cover same parts of source sentence, therefore the one

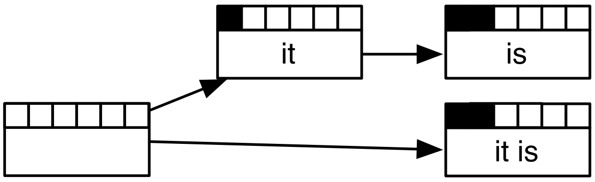


Figure 2.3: Before recombination

Koehn (2009)

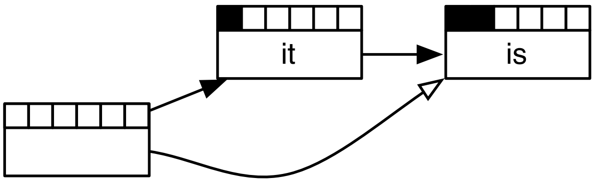


Figure 2.4: After recombination

Koehn (2009)

with lower score could be recombined to reduce search space. Although this technique removes many unnecessary paths from graph, applying recombination doesn't affect overall exponential growth in number of hypothesis.

2.3.5 Pruning

Further reduction of 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 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 main reasons of erroneous translation produced by the process described are *search error* and *model error*. While the first one corresponds to failure of finding correct translation during search, second one refers to initial problems in machine translation models. Assistance tool developed and investigated by us aims to provide users with an option to fix these errors within a CAT system by pointing to erroneous parts and selecting a better paraphrasing option from the list retrieved by the system. In some sense our approach lets user to 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 introduced Computer Aided Translation, discussing how multiple assistance ways provided within systems implementing this concept relate to our novel tool that aims to support translators by providing an ability to fix specified parts of the translation using paraphrasing options. Furthermore, we discussed previous paraphrasing approaches highlighting similarities and difference with technique we use to search paraphrases. And finally a brief overview of phrase-based Statistical Machine Translation was provided to introduce origins of the search graph, data structure that we use as main source for paraphrasing.

Current chapter will provide detailed description of work that was carried out during this project. We start by analysing requirements for paraphrasing service within a CAT system. After that an iterational design and implementation process we applied to develop working version of such service. Major design decision that were made during the work on project 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 were queries sent by users are executed against our data model and results after being dynamically filtered and ranked using various heuristics are returned to user interface. In this section we list main requirements for our approach that were considered during 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. It might be easily attached as a module to the services provided online. Moreover, our service could also 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 that provides post-editing assistance, rather requiring any additional paraphrase corpora. We designed our implementation to support translation output generated by Moses []. 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 RePhraser 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 count doesn't exceed 7. This means that it's important to rank 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 are studied. In case of entire sentences the task is known *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 entire sentence, for example results of translation by using different bilingual parallel corpora. 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 stage.

We also took into account following study [], which analyses usage patterns of another assistance tool, bilingual concordancer where 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 and in practice they motivate expectation of average paraphrasing query length to be similar.

Realtime experience. Another requirement for our interactive tool is providing paraphrasing options as fast as possible. Our tool will not be useful at all if it takes it to return results longer that time user spends on manual paraphrasing. Thus we considered time required to generate paraphrases in our evaluation.

Coexistence with manual editing. While RePhraser 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 and 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. Resulting text file contains list of hypotheses that were considered during the decoding stage. Each line in the file represent such hypothesis. Contents of file is ordered by sentence identifiers, that appear in the beginning of each line indicating 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 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 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 translation option addition. Overall score of current partial translation is expressed as score=-1.497, which derives from a log-probability which is 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 forward=3437 and score fscore=-8.900 to the end of the graph. Original sentence coverage information is available in following 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 hypothesis groups *recombined hypotheses*. That 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 additional attribute recombined=731 indicates to the hypothesis with a better score, with which current hypothesis was recombined.

Another option fetching 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 best path that is being added in case of the switch we described earlier. That is why we prefer using the -output-search-graph option instead.

More details are available at [http://goo.gl/6RYbDK]

3.2.2 Preprocessing data for paraphrasing

For initial testing purposes we used Moses to translate set of 3000 Russian sentences from 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 attributes into a dictionary, which 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.

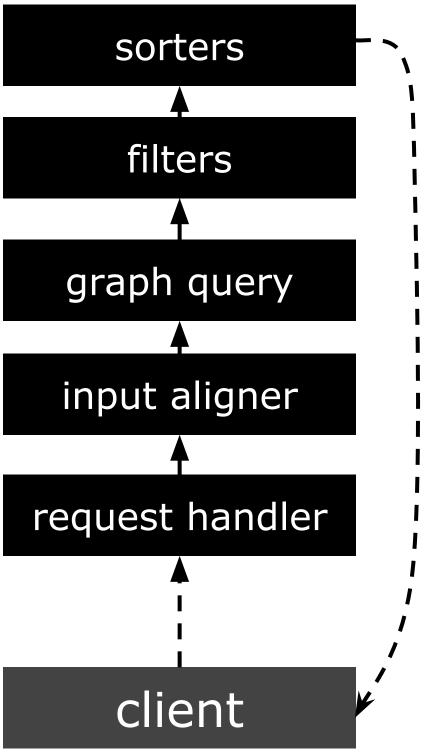
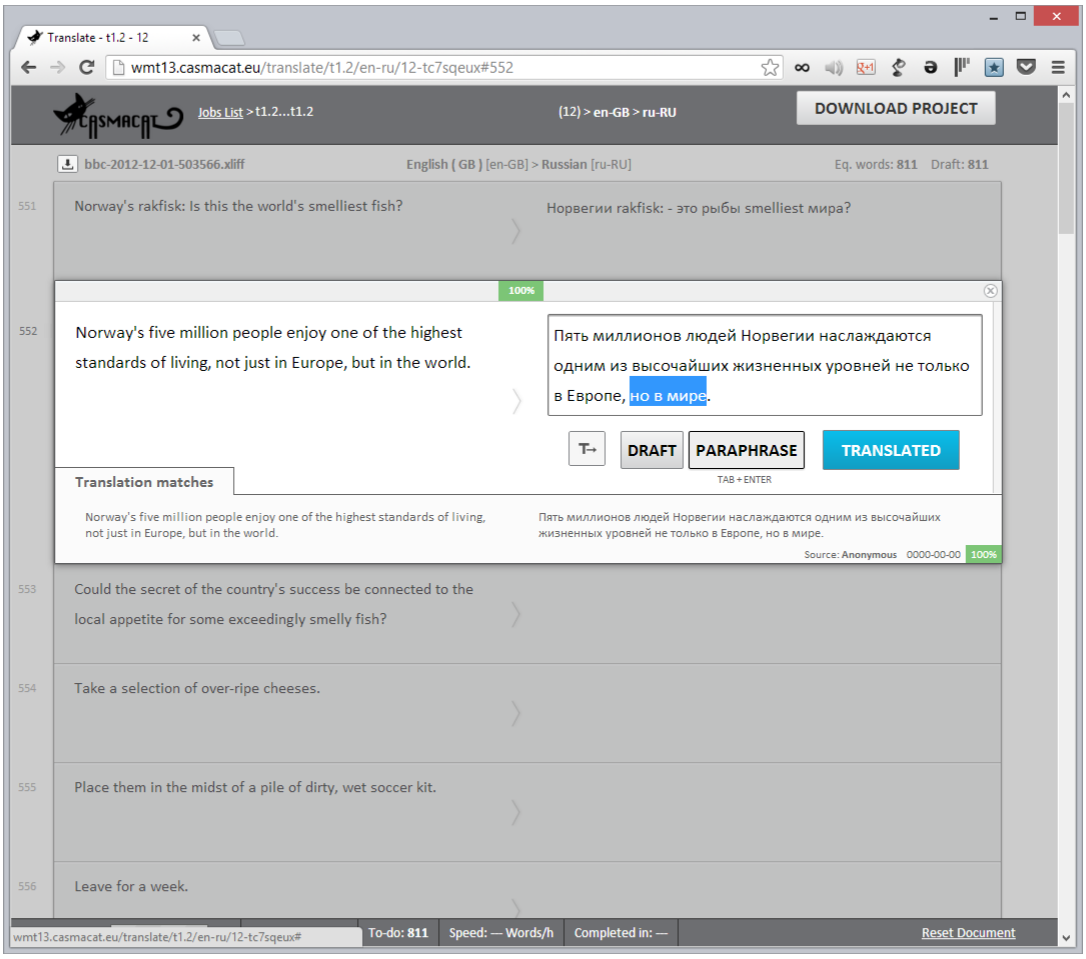


Figure 3.1: Paraphrasing request handling

3.3 Project design outline

In this section we want to introduce high-level structure of the project listing it's main components and their short descriptions. Following sections will provide more details about implementation of these components. As it was mentioned before, our service designed using client-server architecture. While client-side implementation may vary depending on particular CAT software within which paraphrasing service is provided, and integrating our solution with a particular CAT system was out of the scope of current work, we have implemented simple web based mock 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 database graph was reviewed in previous section and it could be executed as an initialisation step triggered after the machine translation for post-editing is produced. Simple sequence of components that are used to process paraphrasing requests are illustrated in Figure 3.1.


Translate - t1.2 - 12
wmt13.casmacat.eu/translate/t1.2/en-ru/12-tc7sqeux#552
CASMACAT
Jobs List > t1.2...t1.2
(12) > en-GB > ru-RU
DOWNLOAD PROJECT
bbc-2012-12-01-503566.xliff
English (GB) [en-GB] > Russian [ru-RU]
Eq. words: 811 Draft: 811
100%
DRAFT
PARAPHRASE
TRANSLATED
TAB + ENTER
Translation matches
Source: Anonymous 0000-00-00 100%
wmt13.casmacat.eu/translate/t1.2/en-ru/12-tc7sqeux#
To-do: 811
Speed: — Words/h
Completed in: —
Reset Document

551 Norway's rakfisk: Is this the world's smelliest fish? — Норвегии rakfisk: - это рыбы smelliest мира?

552 Norway's five million people enjoy one of the highest standards of living, not just in Europe, but in the world. — Пять миллионов людей Норвегии наслаждаются одним из высочайших жизненных уровней не только в Европе, но в мире.

Norway's five million people enjoy one of the highest standards of living, not just in Europe, but in the world. — Пять миллионов людей Норвегии наслаждаются одним из высочайших жизненных уровней не только в Европе, но в мире.

553 Could the secret of the country's success be connected to the local appetite for some exceedingly smelly fish?

554 Take a selection of over-ripe cheeses.

555 Place them in the midst of a pile of dirty, wet soccer kit.

556 Leave for a week.

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 CASMACAT system, 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 user's paraphrasing query is executed against our graph database, it is being aligned with corresponding part original sentence in source language. 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 original sentence parts that correspond to user's selection, fetching original language phrases is not required.

3.3.4 Graph query execution

Having required information about coverage, system executes a query against 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. Results represent a list of unfiltered translation options that are have high probability of being suitable paraphrases to user's selection. Each translation option is paired with data about corresponding hypothesis scores, that are used in further stages of paraphrasing workflow.

3.3.5 Filters, scorers and sorters

Results retrieved from database usually are very redundant and have formatting issues. Thus the next step is filtering them and after that using scoring function to generate final ranking. This step encapsulates the most significant part of 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's query we can execute different sets of filters and sorters. For example, as we will see 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 we described will be provided. We will start by discussing the parts that are involved in paraphrasing directly. Description of each component will also contain problematic cases and their solutions discovered by us. Next we will give a brief overview of client-server interaction implementation.

3.4.1 Implementing input aligner

Input aligner detects coverage of user's selection submitted for paraphrasing. Later 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 of 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 don't 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 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 indicies. It is 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 indicies of corresponding words in translation.

3. **Finding involved intervals**. The final step is merging results of previous two steps to find indicies 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 alignment 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 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 on phrase level and do not provide us with 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's input to match the whole phrases. We also considered one exceptional case, when phrase partial 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`.

Finally, alignment logic gets more complicated in case when user submits selection after manually editing translation. In this case user's input may not exactly match 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 matches based on edit distance. While this practice is dangerous in case of short selections, with multiword inputs we can take into account that words should match parts of 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 may be expressed in terms of Hidden Markov Models (HMM). As these models were not used in core paraphrasing implementation, we will not provide detailed definition of them, which could be found at []. 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 translation output represents an observation. Each observation corresponds to one of the hidden states that are words in user's selection plus special state "none", which means that observation doesn't correspond to any input word. Sensor model for HMM uses string edit distance and other features between observation and states. Transition model is based on the sequence of words in user's input. For our example, probability of the first w_2 and second w_5 occurrences of `us` are calculated as:

$$P(us|w_2) = P(w_2|us)P(us|none) \tag{3.1}$$

$$P(us|w_5) = P(w_5|us)P(us|told) \tag{3.2}$$

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’s 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 doesn’t 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 search graph database. We built 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 attributes available from the search dump file, that we described in Section 3.2.

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

- *Hypotheses should be from the same sentence.* This basic limitation significantly reduces the search space by focusing only of 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 build using parallel corpus. This fact relates our approach to 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 paraphrases ranking.

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 can build queries for each possible separation of original coverage. However the number of possible ways to split coverage into phrases grows 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 also use information about the best next step (`forward` and `fscore`) to detect the most useful grouping options. Here are descriptions of operations that are carried out to retrieve hypotheses that are 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 user's 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 add record it's 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 machine during translation. For our example, two new coverage types are: `|4-4|` and `|4-5|`.

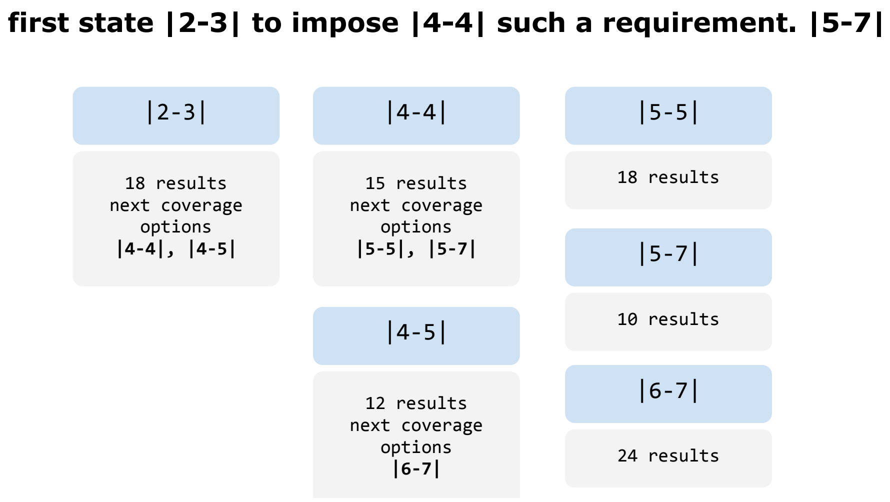


Figure 3.3: Retrieving partial paraphrases with different coverage

- After removing coverage types that are not withing overall coverage of user's selection (in our case |2-7|). We run the procedure described in previous step recursively for each coverage type. Results of all queries are recorded in a dictionary data structure, where key represents coverage type and value contain 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 and 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 is described in Koehn (2009) Section 9.1.2.

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 cost is associated with each translation. For each state the cost of outgoing transitions are high if they are not expected to lead to a good paraphrase, and the cost is low for promising paths. These costs are calculated using various features like string distance, diversity factor and language model score. For each feature there is a *scorer* that assesses it for a given transition and adds the result to the final score. The scorers 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 partials by similarity, function words, 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 by maintaining the overall coverage. The end states of our FSM are the states with coverage that matches the end of the overall coverage. 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 in Figure 3.4.

After constructing a FSM, the next step is to generate 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 the best known ones is *Yen's algorithm* Yen (1971). We used it's 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 for long phrases. 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 order to achieve 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

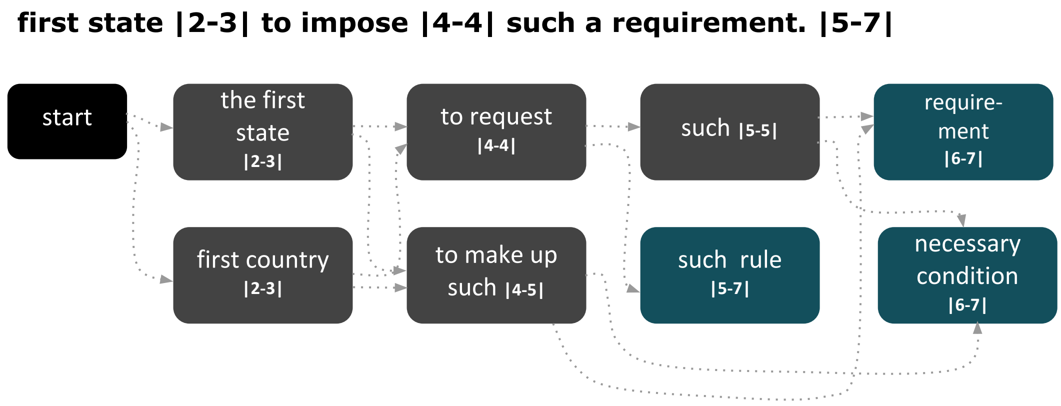


Figure 3.4: Part of a sample finite state machine

small enough to keep paraphrasing decision simple for the user. On this stage filtering and sorting implemented similarly to partial filters and scorers. Again we have an optionally attachable *filters* and *sorters*, that form final list of paraphrasing options. We will discuss existing filters, scorers and sorters in following 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 it's 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 a same format. The groups include: *filters*, *partial filters*, *scorers* 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 partials 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. This way before applying filtering logic, our partial filter will execute another partial filter and the will use it's results as initial data for filtering. Using both constructors makes it possible to dynamically build complex filters. This design solution is the variation of the Decorator pattern, we mentioned before. Following pseudocode illustrates 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 we 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 output it.

`SDPF`: String Distance Partial Filter
For each item in the list we calculate Levenshtein string distance between it and 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's 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. For all our experiments we used $t = 3$, because intuitively filtering items with longer distance we may lose good partial paraphrasing candidates.

FWPF: Function Word Partial Filter

For each item we remove following items if the only difference between two string is in function words. This way in the resulting list there are still function words. 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. Initial score is set to be equal to the score produced by previous score function. And after calculating score with current function, result equals to the average of two scores. A 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 reuses information about the best next step from search graph. It checks if target hypothesis has the same id as `forward` reference for 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.

SDSF: Score Difference Score Function

Function calculates the difference between scores for two states. Output is relative difference score between 0 and 1, calculated as:

$$\min(\frac{score_t}{2score_s}, 1) \tag{3.3}$$

Here $score_t$ is the target score and $score_s$ is the source score. As a result output will be high if as a result of transition search graph score increases, and low otherwise.

`LMSF`: Language Model Score Function
This function uses language model that was used for machine translation to score the result of merging outputs of two hypotheses. In order to get language model score we use Moses 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 use only overall total score.

3.4.4.3 Filters

In contrast with partial filters, regular filters are applied to final list of paraphrasing options. This list is generated using FSM and we still may want to remove paraphrasing candidates that are very similar. 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 and the Function Word partial filters. They are correspondingly named **String Distance Filter** (**`SDRF`**) and **Function Word Filter** (**`FWFF`**). The only novel filter is a score based filter and it's description is provided here:

`SBRF`: Score Based Filter
This simple filter removes item's that have a final FSM transition score which is below a predefined threshold. The filter is used only in cases when number of results it 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 sorters reorder items in the resulting list. Initial list passed to sorters is already sorted by FSM transition 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, this sorter uses language model score for a whole sentence by replacing paraphrased part with a paraphrasing option. This way paraphrase is assessed in context of the surrounding words. Items in the list are rearranged by score in descending order.

`CBDS`: Cluster Based Diversity Sorter
This sorter is based on a heuristic that more diverse results are more useful for user. Indeed, a user searching for paraphrases is likely to be looking for a way to express idea in a different way, rather that trying to fix a minor spelling mistake. To achieve more contrasting results we use simple clustering approach. We group items into clusters based on word distance. The distance is calculated in a way similar to 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 []. Order within clusters 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 different from each other, but at the same time have high scores.

3.5 Summary

Chapter 4

Evaluation

In this chapter we provide detailed description of evaluation process that we carried out for testing variations of our paraphrasing approach. Considering the modular design of our paraphraser it was possible to design multiple experiments by adding and removing certain functionality. In scope of this project we were mainly interested in evaluating usefulness of paraphrasing options provided by our tool in context of translation task. Considering that it wasn't feasible to set up user studies at early stage of the project, we designed an automatic evaluation system that helped us to find out various issues and interesting features about our tool. We will begin this chapter by describing this automatic evaluation framework. At the final stage of project we also carried out a small user study. We will show how results of this study supported our earlier 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 we described in the background. Our main goal was to test if 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 machine translation corrupted sentence and a set of suggested paraphrases for different parts of this sentence. In our evaluation process we treat these suggested paraphrases as 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 most 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 process of test case generation, then we will provide the evaluation results and we will conclude by discussing problems with this type of evaluation that were found by us.

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"
```

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/valil/wmt13-en-ru/evaluation/newstest2013.input.lc.1 > ~/rus_out_1
```

Figure 4.1 illustrates sample original English sentence (a) and it's the 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 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"
```

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
```

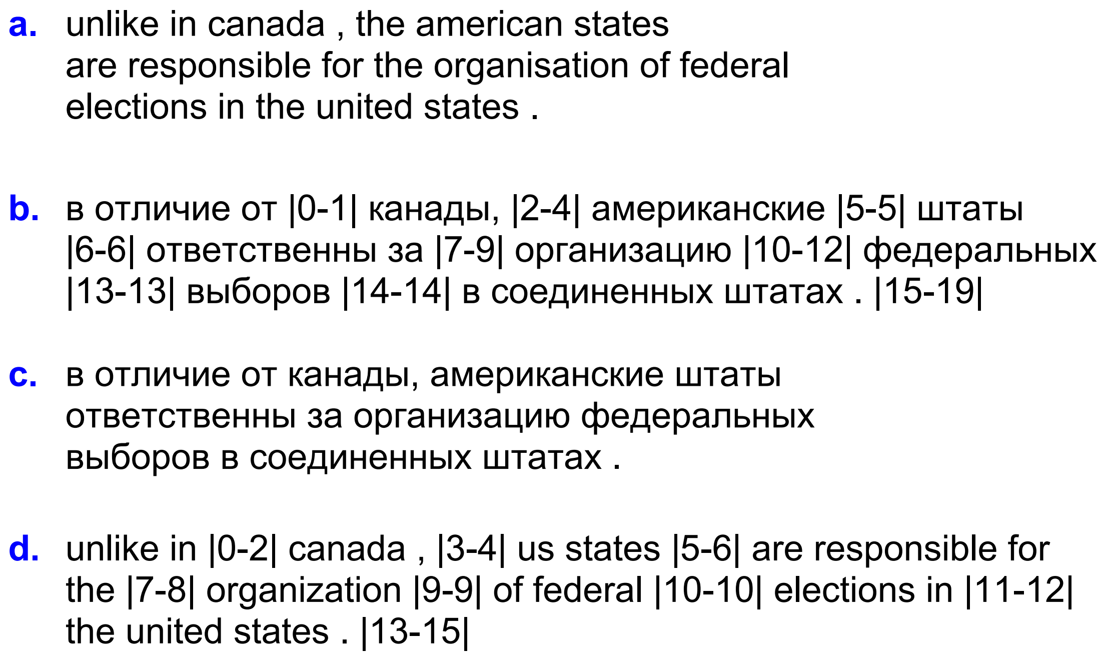
The resulting English sentence is illustrated in Figure 4.1 (d). As we can see it is different from original English input. In this case the difference is in phrases "us states" and "american states". Obviously, original phrase is a better translation and if our paraphraser can suggest it in top 5 results 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 cases repository. To detect corrupted parts that are different from original, 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". We compare this to "us states" and spot the difference. We applied the same procedure for all phrases in final English translation, merging sequential difference intervals into one.

As a result we collected 2139 test cases, that we 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 is contains 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|

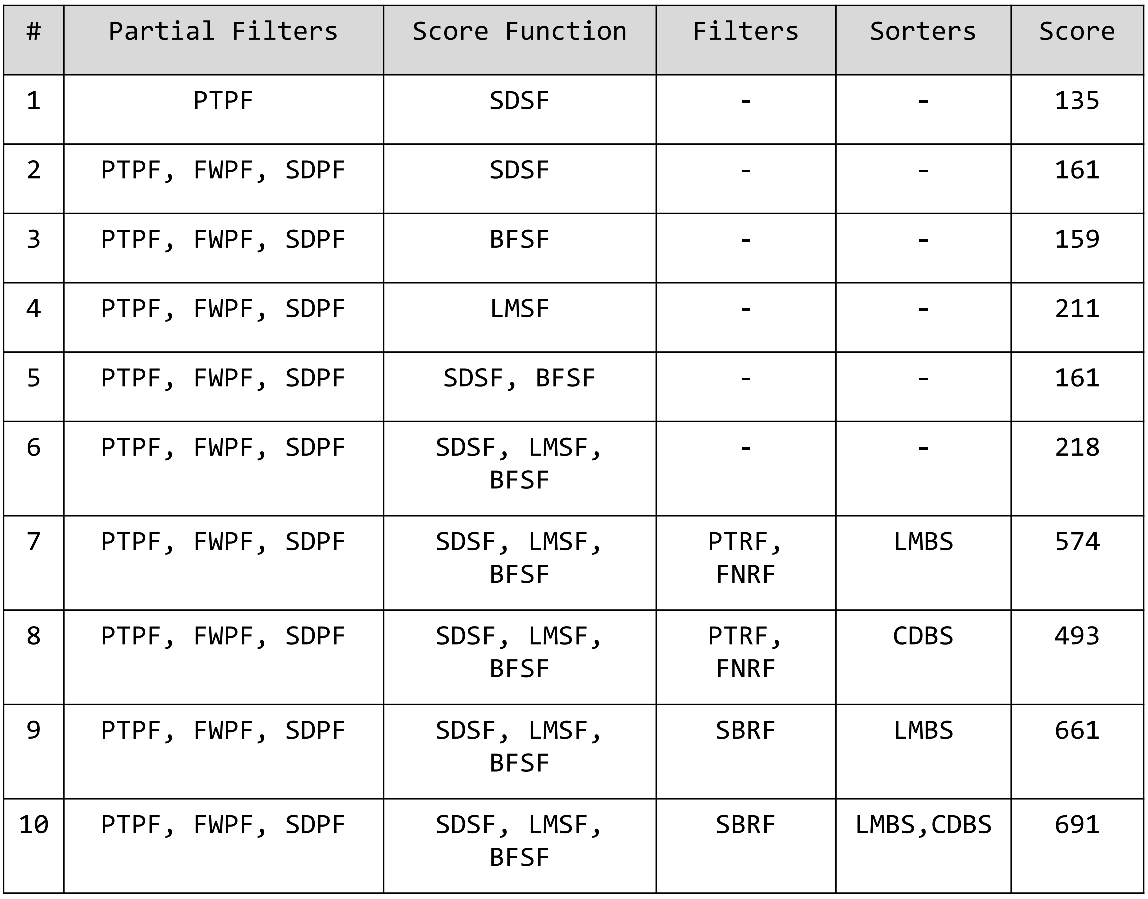
Figure 4.1: Resulting Russian translation

4.1.3 Analysing automatic evaluation results

As we can see from results illustrated in Figure 4.2, we achieve 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. Before sorting final results the best approach also filters them through the Score Based Filter. In contrast, Approach 1 has the worst performance. For this approach we didn't use any filters or sorters. However we used punctuation partial filter and score difference based score function. We consider Approach 1 as baseline approach, because it uses only minimum required functions. Furthermore, we can see that we achieve best effects 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 the 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 automatic evaluation in greater details. This tool lists all test cases per experiment. Each item of the list



#	Partial Filters	Score Function	Filters	Sorters	Score
1	PTPF	SDSF	-	-	135
2	PTPF, FWPF, SDPF	SDSF	-	-	161
3	PTPF, FWPF, SDPF	BFSF	-	-	159
4	PTPF, FWPF, SDPF	LMSF	-	-	211
5	PTPF, FWPF, SDPF	SDSF, BFSF	-	-	161
6	PTPF, FWPF, SDPF	SDSF, LMSF, BFSF	-	-	218
7	PTPF, FWPF, SDPF	SDSF, LMSF, BFSF	PTRF, FNRF	LMBS	574
8	PTPF, FWPF, SDPF	SDSF, LMSF, BFSF	PTRF, FNRF	CDBS	493
9	PTPF, FWPF, SDPF	SDSF, LMSF, BFSF	SBRF	LMBS	661
10	PTPF, FWPF, SDPF	SDSF, LMSF, BFSF	SBRF	LMBS,CDBS	691

Figure 4.2: Automatic evaluation results

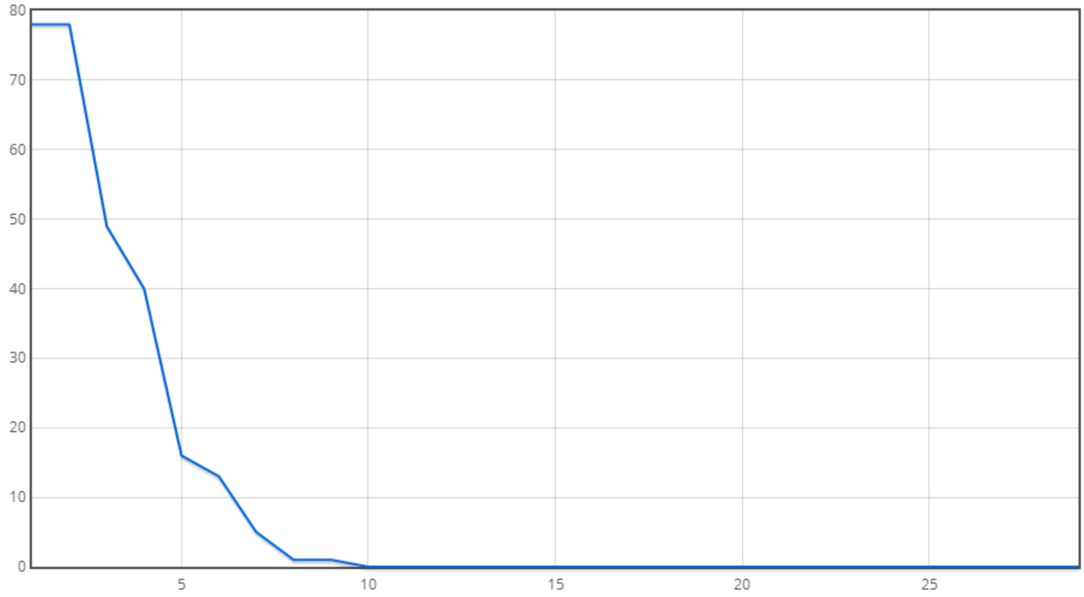


Figure 4.3: Relation between phrase size and paraphrasing performance

X axis: number of words in phrase, Y axis: percentage of successful cases

is red in case of failure or green in case of success. We placed “Details” button for 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 is available on the project website and was implemented using JavaScript.

Using results visualisation tool we detected multiple problems with this way of 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, that despite not matching the original natural language English phrases, should be considered as success cases for paraphraser. Indeed, 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.

4.2 Manual evaluation

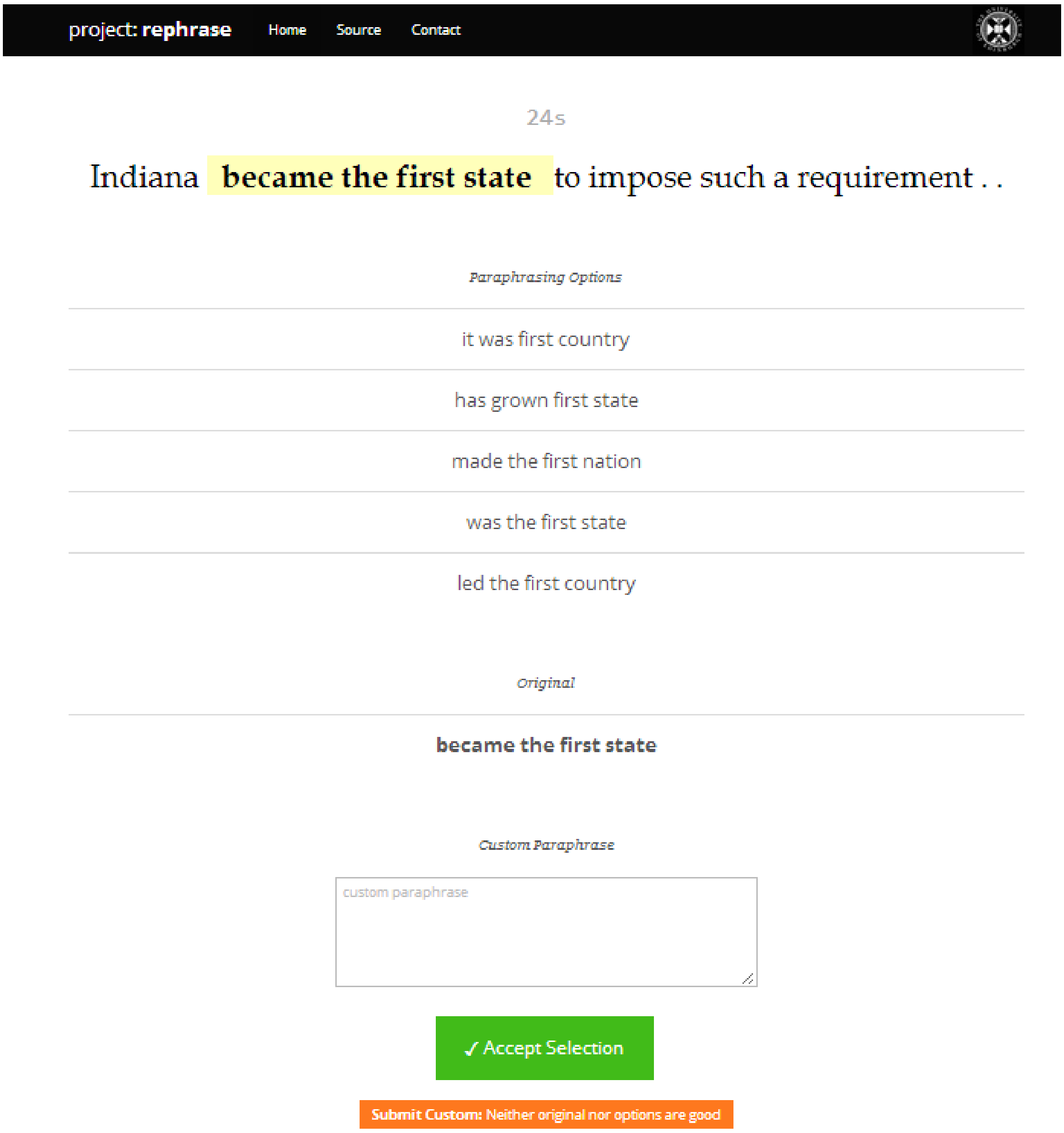
During this project we also developed a web based interactive evaluation tool. This tool provides a user with an interface for manual evaluation of paraphrasing results. It reuses automatic evaluation test cases, displaying machine translation of the sentence and highlighting an area that is considered 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 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 reused in our JavaScript templates.

4.2.1 Manual evaluation methodology

Considering results of automatic evaluation, we picked three out of ten tested approaches for the user study. These approaches are 1, 7 and 10. From 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 by their initials in the following way: `HG`, `OM`, `NH` and `RM`. For each of the three approaches each participant submitted his decisions the same 50 test cases. We also considered the case when user comes across a test case that he already assessed for previous approach and his desired paraphrase selection already is in top 5 for current approaches. We automatically skipped this test cases, considering them successful.



24s

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

Figure 4.4: Manual evaluation tool

#	HG	OM	NH	RM	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 tool

4.2.2 Analysing manual evaluation results

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

Bibliography

Callison-Burch, C. (2007). Paraphrasing and Translation Doctor of Philosophy School of Informatics University of Edinburgh.

Gamma, E., Vlissides, J., Helm, R., and Johnson, R. (1995). Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49:120.

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.

Macklovitch, E., Lapalme, G., and Gotti, F. (2008). TransSearch : What are translators looking for ?

Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716.