

PBML



The Prague Bulletin of Mathematical Linguistics

NUMBER 100 OCTOBER 2013

EDITORIAL BOARD

Editor-in-Chief

Eva Hajičová

Editorial staff

Matěj Korvas

Ondřej Bojar

Martin Popel

Editorial board

Nicoletta Calzolari, Pisa

Walther von Hahn, Hamburg

Jan Hajič, Prague

Eva Hajičová, Prague

Erhard Hinrichs, Tübingen

Aravind Joshi, Philadelphia

Philipp Koehn, Edinburgh

Jaroslav Peregrin, Prague

Patrice Pognan, Paris

Alexandr Rosen, Prague

Petr Sgall, Prague

Hans Uszkoreit, Saarbrücken

Published twice a year by Charles University in Prague

Editorial office and subscription inquiries:

ÚFAL MFF UK, Malostranské náměstí 25, 118 00, Prague 1, Czech Republic

E-mail: pbml@ufal.mff.cuni.cz

ISSN 0032-6585



The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013

CONTENTS

Editorial	5
------------------	---

Articles

Makefiles for Moses	9
<i>Ulrich Germann</i>	
QuEst — Design, Implementation and Extensions of a Framework for Machine Translation Quality Estimation	19
<i>Kashif Shah, Eleftherios Avramidis, Ergun Biçici, Lucia Specia</i>	
MTMonkey: A Scalable Infrastructure for a Machine Translation Web Service	31
<i>Aleš Tamchyna, Ondřej Dušek, Rudolf Rosa, Pavel Pecina</i>	
DIMwid — Decoder Inspection for Moses (using Widgets)	41
<i>Robin Kurtz, Nina Seemann, Fabienne Braune, Andreas Maletti</i>	
morphogen: Translation into Morphologically Rich Languages with Synthetic Phrases	51
<i>Eva Schlinger, Victor Chahuneau, Chris Dyer</i>	
RankEval: Open Tool for Evaluation of Machine-Learned Ranking	63
<i>Eleftherios Avramidis</i>	
XenC: An Open-Source Tool for Data Selection in Natural Language Processing	73
<i>Anthony Rousseau</i>	

COSTA MT Evaluation Tool:	83
An Open Toolkit for Human Machine Translation Evaluation <i>Konstantinos Chatzitheodorou, Stamatis Chatzistamatis</i>	
Open Machine Translation Core:	91
An Open API for Machine Translation Systems <i>Ian Johnson</i>	
CASMACAT: An Open Source Workbench for Advanced Computer Aided Translation	101
<i>Vicent Alabau, Ragnar Bonk, Christian Buck, Michael Carl, Francisco Casacuberta, Mercedes García-Martínez, Jesús González, Philipp Koehn, Luis Leiva, Bartolomé Mesa-Lao, Daniel Ortiz, Herve Saint-Amand, Germán Sanchis, Chara Tsoukala</i>	
Sequence Segmentation by Enumeration: An Exploration <i>Steffen Eger</i>	113
Instructions for Authors	133



The Prague Bulletin of Mathematical Linguistics

NUMBER 100 OCTOBER 2013

EDITORIAL

50 years of The Prague Bulletin of Mathematical Linguistics

Half a century of the existence of a scientific journal is quite a long life span, especially if one takes into account the specificity of the political development and turbulences in the country of origin, namely Czech Republic (former Czechoslovakia), and the branch of science, namely computational (mathematical) linguistics. And yet, it was fifty years ago, in 1964, when the first issue of The Prague Bulletin of Mathematical Linguistics, published by Charles University in Prague, appeared, with 3 full papers and 5 review articles, in an edition of 250. The ambitions of the editor-in-chief (Petr Sgall, still participating in the present-day editorial board) and the editorial board (a logician Karel Berka, a general linguist Pavel Novák and a specialist in quantitative linguistics Marie Těšitelová; to our deep sorrow, none of the three can celebrate with us today) as declared in the first Editorial were rather modest but also rather urgent at the time: to provide a forum for Czech researchers in the newly developing field of mathematical linguistics and its applications to inform the international community about their research activities, results and standpoints. As the university department that was responsible for the publication of PBML included in its name the attribute “algebraic linguistics”, the Editorial also referred to its orientation using this attribute (borrowed from Y. Bar-Hillel) to distinguish the new trend in linguistics from the at that time already well-established field of quantitative (called also statistical, sic!) linguistics. The editors expressed their appreciation of N. Chomsky’s contribution to theoretical linguistics esp. in connection with the formal specification of language by means of generative system and the assignment of structural characteristics to sentences and emphasized the possibility offered by such an approach to compare different types of grammars by means of usual mathematical methods. However, they also warned that there are some difficulties concerning the mathematical formulation of transformational grammar and its linguistic interpretation and suggested that it is desirable to have another alternative of the generative description of language. They referred to classical Praguian understanding of the relation of form and function and the multilevel approach on the one side, and to such (at that time) contemporary researchers as H. B. Curry, H. Putnam, S. K. Shaumjan or I. I. Revzin on the other. It should be noticed that already in this very brief Editorial the possibility to use a dependency rather than a constituency based account of syntactic relations

was mentioned, as well as the importance of including semantic considerations into linguistic description (as well as into possible applications, which, at that time, mostly concerned machine translation).

It should be remembered that this Editorial was written at the beginning of 1964, before the appearance of Katz and Postal's monograph on an integrated theory of linguistic description and one year before the publication of Chomsky's *Aspects* and his idea of the difference between deep and surface structure, not to speak about the split within transformational grammar in the years 1967–1969 into the so-called interpretative and generative semantics. In a way, the contents of the Editorial somehow signaled the appearance of the alternative generative approach of formal description of language as proposed in mid-sixties by Petr Sgall and as developed further by his collaborators and pupils, i.e. the so-called Functional Generative Description (FGD). There are three distinguishing features of this theoretical approach, namely (i) a multi-level (stratificational) organization of linguistic description, with the underlying syntactic level (called tectogrammatical, using Putnam's terminological distinction between pheno- and tecto-grammatics) as its starting point, (ii) a dependency account of syntactic relations with valency as its basic notion, and (iii) the inclusion of the description of the topic-focus articulation (TFA, now commonly referred to as the information structure of the sentence) into the underlying level of the formal description of language. In the years to follow, FGD was not only used as the theoretical framework for the description of multifarious linguistic phenomena (not only of Czech, but also in comparative studies of Czech and English, or other, mostly Slavonic languages), but also as a basis for the formulation of an annotation scheme for corpora applied in the so-called Prague Dependency Treebank 30 years later.

Back to the history of PBML. Its appearance in 1964 actually indicates that the political situation in the mid-sixties though still very tough, intolerable and difficult to live through was not so strictly adversative to some till then unimaginable movements in cultural and scientific life, especially if some parallel tendencies could be found in Soviet Russia. It was in the same year, September 18–22, 1964, when a first (rather small) international meeting on computational linguistics took place in Prague, called Colloquium on Algebraic Linguistics, in which such prominent scholars as J. J. Ross and E. S. Klima from the U.S., M. Bierwisch, J. Kunze and H. Schnelle from Germany, J. Mey from Norway, H. Karlgren and B. Brodda from Sweden, B. Vauquois from France, F. Papp, F. Kiefer and L. Kálmár from Hungary participated; altogether there were 35 participants from abroad and tens of interested mostly young scholars from Czechoslovakia. (One should be aware of the fact that this was one year before the start of the regular international meetings on computational linguistics later known as COLING (organized by the International Committee on Computational Linguistics) and the Annual ACL conferences organized by the Association for Computational Linguistics.) However, the situation dramatically changed soon (though not immediately, but with a delay of a year or two) after the Russian invasion to Czechoslovakia in 1968. This change was reflected also in the position of the research team of mathe-

matical linguistics at the Faculty of Arts at Charles University in Prague: in 1970 the team lost the status of a department, in 1972 the Head of the Laboratory Petr Sgall was threatened to have to leave the University and a similar fate was expected to be faced by all of the members. Thanks to the consistence and solidarity of the team and also to the help of our colleagues at the Faculty of Mathematics and Physics all the members of the team found an “asylum” at different departments (though not as a laboratory of its own) at this ideologically less strictly watched faculty.

At that point, it was clear to us that the very existence of the Prague Bulletin was in a great danger. And again, solidarity was a crucial factor: one of the original Editorial Board members, the well-known logician prof. Karel Berka, the only member of the Communist Party in the Board and actually not a computational linguist, took over the initiative and actively fought for the continuation of the Bulletin. Its existence was really extremely important – it helped to keep us in contact with the international scene, not only by informing our colleagues abroad about our work but also, maybe even more importantly at that time, to have something to offer “in exchange” for publications and journals published abroad which were – due to currency restrictions – not otherwise available in our country. In this way, Czech(oslovak) computational linguistics has never lost contacts with the developments in the field. One of the remarkable sources of information, for example, were the mimeographed papers, PhD theses and pre-publications produced and distributed by the Indiana University Linguistics Club at Bloomington University, Indiana, which we were receiving free of charge, not “piece for piece” (which would mean only two papers in a year, since PBML was a bi-annual journal), but tens of papers for one PBML issue. Thanks to the solidarity and friendliness of our colleagues at most different universities and research institutions abroad, a similar exchange policy was in existence for more than two decades, even between the PBML publishers and Editorial Boards or publishers of some regular scientific journals.

In the course of the fifty years of its existence, our journal has faced not only difficulties but also some favorable developments. The journal has become more international: the contents is no longer restricted to contributions of Czech scholars, as originally planned, the Editorial Board has undergone several changes the most important of which was introduced in June 2007 (PBML 87), when the Editorial Board was enlarged by prominent scholars of the field from different geographical areas as well as domains of interest, and the review process was made more strict by having at least one reviewer for each submission from abroad. At the same time, we started to make the individual issues available on the web and also the format of the journal and its graphical image has considerably improved. Starting from PBML 89, all articles have assigned DOI identifiers and they are published also via the Versita (De Gruyter) open access platform.

The thematic scope of PBML is also rather broad; the Editorial Board is open to publish papers both with a theoretical as well as with an application orientation, as testified by the fact that since 2009 (PBML 91) we publish regularly the papers accepted

for presentation at the regular Machine Translation Marathon events organized by a series of EU-funded projects: EuroMatrix, EuroMatrixPlus and now MosesCore. We are most grateful to the group of reviewers of the Marathon event who present their highly appreciated comments on the tools described in the papers. PBML has thus become one of a very few journals that provide a traditional scientific credit for rather practical outcomes: open-source software, which can be employed in further research and often also outside of academia right away.

We are convinced that in the course of the fifty years of its existence, The Prague Bulletin of Mathematical Linguistics has developed into a fully qualified member of the still growing family of journals devoted to many-sided issues of computational linguistics and as such will provide an interesting and well-received forum for all researchers irrespective of their particular specialization, be they members of the theoretically or application oriented community.

Eva Hajičová, Petr Sgall and Jan Hajič

{hajicova,sgall,hajic}@ufal.mff.cuni.cz



Makefiles for Moses

Ulrich Germann

University of Edinburgh

Abstract

Building MT systems with the *Moses* toolkit is a task so complex that it is rarely done manually. Over the years, several frameworks for building, running, and evaluating *Moses* systems have been developed, most notably the *Experiment Management System* (EMS). While EMS works well for standard experimental set-ups and offers good web integration, designing new experimental set-ups within EMS is not trivial, especially when the new processing pipeline differs considerably from the kind EMS is intended for. In this paper, I present M4M (*Makefiles for Moses*), a framework for building and evaluating *Moses* MT systems with the *GNU Make* utility. I illustrate the capabilities by a simple set-up that builds and compares two different systems with common resources. This set-up requires little more than putting training, tuning and evaluation data into the right directories and running *Make*.¹ The purpose of this paper is twofold: to guide first-time users of *Moses* through the process of building baseline MT systems, and to discuss some lesser-known features of the *Make* utility that enable the MT practitioner to set up complex experimental scenarios efficiently. M4M is part of the *Moses* distribution.

1. Introduction

The past fifteen years have seen the publication of numerous open source toolkits for statistical machine translation (SMT), from word alignment of parallel text to decoding, parameter tuning and evaluation (Och and Ney, 2003; Koehn et al., 2007; Li et al., 2009; Gao and Vogel, 2008; Dyer et al., 2010, and others). While all these tools greatly facilitate SMT research, building actual systems remains a tedious and complex task. Training, development and testing data have to be preprocessed, cleaned

¹For the sake of convenience, I use *Make* to refer to *GNU Make* in this paper. *GNU Make* provides a number of extensions not available in the original *Make* utility.

up and word-aligned. Language and translation models have to be built, and system parameters have to be tuned for optimal performance. Some of these tasks can be performed in parallel. Some can be parallelized internally by a split-and-merge approach. Others need to be executed in sequence, as some build steps depend on the output of others.

There are generally three approaches to automating the build process. The first approach is to use **shell scripts** that produce a standard system setup. This is the approach taken in *Moses for Mere Mortals*.² This approach works well in a production scenario where there is little variation in the setup, and where systems are usually built only once. In a research scenario, where it is typical to pit numerous systems variations against one another, this approach suffers from the following drawbacks.

- Many of the steps in building SMT systems are computationally very expensive. Word alignment, phrase table construction and parameter tuning can each easily take hours, if not days, especially when run without parallelization. It is therefore highly desirable *not* to recreate resources unnecessarily. Building such checks into regular shell scripts is possible but tedious and error-prone.
- When the build process fails, it can be hard to determine the exact point of failure.
- Parallelization, if desired, has to be hand-coded.

The second approach is to write a **dedicated build system**, such as the *Experiment Management System* (EMS) for *Moses* (Koehn, 2010), or *Experiment Manager* (*Eman*), a more general framework for designing, running, and documenting scientific experiments (Bojar and Tamchyna, 2013).

EMS was designed specifically for *Moses*. It is capable of automatically scheduling independent tasks in parallel and includes checks to ensure that resources are only (re)created when necessary. EMS works particularly well for setting up a standard baseline system and then tweaking its configuration manually, while EMS keeps track of the changes and records the effect that each tweak has on overall system performance. In its job scheduling capabilities, EMS is reminiscent of generic build systems such as *Make*. In fact, the development of EMS is partly due to perceived shortcomings of *Make* (P. Koehn, personal communication), some of which we will address later on.

As a specialized tool that implements a specific way of running *Moses* experiments, EMS has a few drawbacks, too. Experimental setups that stray from the beaten path can be difficult to specify in EMS. In addition, the point of failure is not always easy to find when the system build process crashes, especially when the build failure is due to errors in the EMS configuration file.

²http://en.wikipedia.org/wiki/Moses_for_Mere_Mortals,
<https://code.google.com/p/moses-for-mere-mortals>

Eman (Bojar and Tamchyna, 2013) also has its roots in SMT research but is designed as a general framework for running scientific experiments. Its primary objectives are to avoid unnecessary recreation of intermediate results, and to ensure that all experiments are replicable by preserving and thoroughly documenting all experimental parameters and intermediate results. To achieve this, *Eman* has a policy of never overwriting or re-creating existing files. Instead, *Eman* clones and branches whenever an experiment is re-run. Due to its roots, *Eman* comes with a framework for running standard SMT experiments.

The third approach is to rely on **established generic build systems**, such as the *Make* utility. *Make* has the reputation of being arcane and lacking basic features such as easy iteration over a range of integers, and much of this criticism language is indeed justified — *Make* is not for the faint-of-heart. On the other hand, it is a tried-and-tested power tool for complex build processes, and with the help of some of the lesser-known language features, it can be extremely useful also in the hands of the MT practitioner.

This article is foremost and above all a tutorial on how to use *Make* for building and experimenting with *Moses* MT systems. It comes with a library of *Makefile* snippets that have been included in the standard *Moses* distribution.³

2. Makefile Basics

While inconveniently constrained in some respects, the *Make* system is very versatile and powerful in others. In this section I present the features of *Make* that are the most relevant for using *Make* for building *Moses* systems.

2.1. Targets, Prerequisites, Rules, and Recipes

Makefile rules consist of a *target*, usually a file that we want to create, *prerequisites* (other files necessary to create the target), and a *recipe*: the sequence of shell commands that need to be run to create the target. The target is (re-)created when a file of that name does not exist, or if any of the prerequisites is missing or younger than the target itself. Prior to checking the target, *Make* recursively checks all prerequisites. The relation between target and prerequisite is called a *dependency*.

Makefile rules are written as follows.

```
target: prerequisite(s)
    commands to produce target from prerequisite(s)
```

Note that each line of the recipe must be indented by a single tab. Within the recipe, the special variables `$$`, `$(`, `^`, and `$|` can be used to refer to the target, the first normal prerequisite, the entire list of normal prerequisites, and the entire list of *order-only* prerequisites, respectively.

³<https://github.com/moses-smt/mosesdecoder>; *Makefiles for Moses* is located under `contrib/m4m`

In addition to regular prerequisites, prerequisites can also be specified as *order-only* prerequisites. Order-only prerequisites only determine the order in which rules are applied, but the respective target is not updated when the prerequisite is younger than the target. Order-only dependencies are specified as follows (notice the bar after the colon).

```
target: | prerequisite(s)
    commands to produce target from prerequisite(s)
```

Makefiles for Moses uses order-only dependencies extensively; it is a safe-guard against expensive resource recreation should a file time stamp be changed accidentally, e.g. by transferring files to a different location without preservation of the respective time stamps.

A number of special built-in targets, all starting with a period, carry special meanings. Files listed as prerequisites of these targets are treated differently from normal files. In the context of this work, the following are important.

.INTERMEDIATE: Intermediate files are files necessary only to create other targets but not important for the final system. If an intermediate file listed as the prerequisite of other targets does not exist, it is created only if the target needs to be (re)created. Declaring files as intermediate allows us to remove files that are no longer needed without triggering the recreation of dependent targets when *Make* is run again.

.SECONDARY: *Make* usually deletes intermediate files when they are no longer required. Files declared as secondary, on the other hand, are never deleted automatically by *Make*. Especially in a research setting we may want to keep certain intermediate files for future use, without having to recreate them when they are needed again. The combination of **.INTERMEDIATE** and **.SECONDARY** give us control over (albeit also the burden of management of) if and when intermediate files are deleted.

2.2. Pattern Rules

Pattern rules are well-known to anyone who uses *Make* for compiling code. The percent symbol serves as a place holder that matches any string in the target and at least one prerequisite. For example, the pattern rule

```
crp/trn/pll/tok/%.de.gz: | crp/trn/pll/raw/%.de.gz
    zcat $< | tokenize.perl -l de | gzip > $@
```

will match any target that matches the pattern `crp/trn/pll/tok/*.de.gz`, check for the existence of a file of the same name in the directory `crp/trn/pll/raw` and execute the shell command

```
zcat $< | tokenize.perl -l de | gzip > $@
```

2.3. Variables

Make knows two ‘flavors’ of variables. By default, variables are expanded *recursively*. Consider the following example. Unlike variables in standard Unix shells, parentheses or braces around the variable name are mandatory in *Make* when referencing a variable.⁴

```
a = 1
b = $(a)
a = 2
all:
    echo $(b)
```

In most conventional programming languages, the result of the expansion of `$(b)` in the recipe would be 1. Not so in *Make*: what is stored in the variable is actually a reference to `a`, not the value of `$(a)` at the time of assignment. It is only when the value is needed in the recipe that each variable reference is recursively replaced by its value at that (later) time.

On the other hand, *simply expanded* variables expand their value at the time of assignment. The flavor of variable is determined at the point of assignment. The operator `'='` (as well as the concatenation operator `'+='` when used to create a new variable) creates a recursively expanded variable; simply expanded variables are created with the assignment operator `':='`.

Multi-line variables can be defined by sandwiching them between the `define` and `endef` keywords, e.g.

```
define tokenize

$(1)/tok/%.$(2).gz: | $(1)/raw/%.$(2).gz
    zcat $$< | tokenize.perl -l $(2) | gzip > $$@

endef
```

Notice the variables `$(1)` and `$(2)` as well as the escaping of the variables `$<` and `$$` by double `$$`. The use of the special variables `$(1)`, ..., `$(9)` turns this variable into a user-defined function. The blank lines around the variable content are intentional to ensure that the target starts at the beginning of a new line and the recipe is terminated by a new line during the expansion by `$(eval $(call ...))` below.

The call syntax for built-in *Make* functions is as follows.

```
$(function-name arg1,arg2,...)
```

⁴Except variables with a single-character name.

User-defined functions are called via the built-in *Make* function `call`. The value of

```
$(call tokenize,crp/trn/pll,de)
```

is thus

```
crp/trn/pll/tok/%.de.gz: | crp/trn/pll/raw/%.de.gz
zcat $< | tokenize.perl -l de | gzip > $@
```

Together with the built-in *Make* functions `foreach` (iteration over a list of space-separated tokens) and `eval` (which inserts its argument at the location where it is called in the Makefile), we can use this mechanism to programmatically generate *Make* rules on the fly and in response to the current environment. For example,

```
directories := $(shell find -L crp -type d -name raw)
$(foreach d,$(directories:%/raw=%),\
$(foreach l,de en,\
$(eval $(call tokenize,$(d),$(l)))))
```

creates tokenization rules for the languages `de` and `en` for all subdirectories in the directory `crp` that are named `raw`. The substitution reference `$(directories:%/raw=%)` removes the trailing `/raw` on each directory found by the shell call to `find`.

3. Building Systems and Running Experiments

3.1. A Simple Comparison of Two Systems

With these preliminary remarks, we are ready to show in Fig. 1 how to run a simple comparison of two phrase-based *Moses* systems, using mostly tools included in the *Moses* distribution. For details on the M4M modules used, the reader is referred to the actual code and documentation in the M4M distribution. The first system in our example relies on word alignments obtained with `fast_align`⁵ (Dyer et al., 2013); the second uses `mgiza++` (Gao and Vogel, 2008). Most of the functionality is hidden in the M4M files included by the line

```
include ${MOSES_ROOT}/contrib/m4m/modules/m4m.m4m
```

The experiment specified in this Makefile builds the two systems, tunes each five times on each tuning set (with random initialization), and computes the BLEU score for each tuning run on each of the data sets in the evaluation set.

The design goal behind the setup shown is to achieve what I call the washing machine model: put everything in the right compartment, and the machine will automatically process everything in the right order. There is a standard directory structure that determines the role of the respective data in the training process, shown in Table 1.

⁵https://github.com/clab/fast_align

```

MOSES_ROOT = ${HOME}/code/moses/master/mosesdecoder
MGIZA_ROOT = ${HOME}/tools/mgiza
fast_align = ${HOME}/bin/fast_align
# L1: source language; L2: target language
L1 = de
L2 = en
WDIR = ${CURDIR}

include ${MOSES_ROOT}/contrib/m4m/modules/m4m.m4m

# both systems use the same language model
L2raw := $(wildcard ${WDIR}/crp/trn/*/raw/*.${L2}.gz)
L2data := $(subst /raw/,/cased/,${L2trn})
lm.order = 5
lm.factor = 0
lm.lazy = 1
lm.file = ${WDIR}/lm/${L2}.5-grams.kenlm
${lm.file}: | ${L2data}
$(eval $(call add_kenlm,${lm.file},${lm.order},${lm.factor},${lm.lazy}))
.INTERMEDIATE: ${L2data}

# for the first system, we use fast_align
word-alignment = fast
system = ${word-alignment}-aligned
ptable = model/tm/${system}.${L1}-${L2}
dtable = model/tm/${system}.${L1}-${L2}
$(eval $(call add_binary_phrase_table,0,0,5,${ptable}))
$(eval $(call add_binary_reordering_table,0,0,8,\
    wbe-mslr-bidirectional-fe-allff,${dtable},${ptable}))
$(eval $(call create_moses_ini,${system}))
SYSTEMS := $(system)

# for the second system, we use mgiza
word-alignment = giza
$(eval $(clear-ptables))
$(eval $(clear-dtables))
$(eval $(call add_binary_phrase_table,0,0,5,${ptable}))
$(eval $(call add_binary_reordering_table,0,0,8,\
    wbe-mslr-bidirectional-fe-allff,${dtable},${ptable}))
$(eval $(call create_moses_ini,${system}))
SYSTEMS += $(system)
ifdef tune.runs
EVALUATIONS :=
$(eval $(tune_all_systems))
$(eval $(bleu_score_all_systems))
all: ${EVALUATIONS}
    echo EVALS ${EVALUATIONS}
else
all:
    $(foreach n,$(shell seq 1 5),${MAKE} tune.runs="$n_$n";)
endif

```

Figure 1. Makefile for a simple baseline system. All the details for building the system are handled by M4M.

<code>crp/trn/pll/</code>	parallel training data
<code>crp/trn/mno/</code>	monolingual training data
<code>crp/dev/</code>	development data for parameter tuning
<code>crp/tst/</code>	test sets for evaluation
<code>model/tm</code>	phrase tables
<code>model/dm</code>	distortion models
<code>model/lm</code>	language models
<code>system/tuned/tset/n/moses.ini</code>	result of tuning system <i>system</i> on tuning set <i>tset</i> (n-th tuning run)
<code>system/eval/tset/n/eset.*</code>	evaluation results for test set <i>eset</i> , translated by system <i>system/tuned/tset/n/moses.ini</i>

Table 1. Directory structure for standard M4M setups

3.2. Writing Modules

The bulk of the system building and evaluation work is done by the various M4M modules. While an in-depth discussion of all modules is impossible within the space limitations of this paper, a few points are worth mentioning here.

One of the inherent risks in using build systems is that two independent concurrent build runs with overlapping targets may interfere with one another, overwriting each other's files. In deviation from the usual philosophy of build systems — recreate files when their prerequisites change — M4M adopts a general policy of only creating files when they do not exist, never recreating them. It is up to the user to first delete the files that they do want to recreate. To prevent concurrent creation of the same target, we adopt the following lock/unlock mechanism.

```

define lock
mkdir -p ${@D}
test ! -e $@
mkdir $@.lock
echo -n "Started_at_$(shell_date)_ " > $@.lock/owner
echo -n "by_process_$(shell_echo $$PPID)_ " >> $@.lock/owner
echo "on_host_$(shell_hostname)" >> $@.lock/owner
endef

define unlock
rm $@.lock/owner
rmdir $@.lock
endef

```

The first line of the lock mechanism ensures that the target's directory exists. The second line triggers an error when the target already exists. Recall that our policy is to never re-create existing files. The third line creates a semaphore (directory creation is an atomic file system operation). When invoked without the `-p` parameter, `mkdir`

will refuse to create a directory that already exists. The logging information added in the fourth and subsequent lines is helpful in error tracking. It allows us to determine easily which process created the respective lock and check if the process is still running.

Another risk is that partially created target files may falsely be interpreted as fully finished targets, either due to concurrent *Make* runs with overlapping targets, or due to a build failure in an earlier run. (Normally, *Make* deletes the affected target if the underlying recipe fails. However, we disabled this behavior by declaring all files `.SECONDARY`.) We can address this issue by always creating a temporary target under a different name and renaming that to the proper name upon successful creation. The pattern for a module definition thus looks as follows.

```
target: prerequisite
    $(lock)
    create-target > $@_
    mv $@_ $@
    $(unlock)
```

4. Conclusion

I have presented *Makefiles for Moses*, a framework for building and evaluating *Moses* MT system within the *GNU Make* framework. The use of the `eval` function in combination with custom functions allows us to dynamically create *Make* rules for multiple systems in the same Makefile, beyond the limitations of simple pattern rules.

A simple but effective semaphore mechanism protects us from the dangers of running multiple instances of *Make* over the same data. By using order-only dependencies and `.INTERMEDIATE` statements, we can specify a build system that creates resources only once, and allows for the removal of intermediate files that are no longer needed, without *Make* recreating them when run again.

Make's tried-and-tested capabilities for parallelization in the build process are fully available.

While *Makefiles for Moses* lacks the bells and whistles of EMS particularly with respect to progress monitoring and web integration of the experimental results, it offers greater flexibility in experimental design, especially with respect to scriptability of system setup.

5. Acknowledgements

The work described in this paper was performed as part of the following projects funded under the European Union's Seventh Framework Programme for Research (FP7): *Accept* (grant agreement 288769), *Matecat* (grant agreement 287688), and *Cas-macat* (grant agreement 287576).

Bibliography

- Bojar, Ondřej and Aleš Tamchyna. The design of Eman, an experiment manager. *Prague Bulletin of Mathematical Linguistics*, 99:39–58, April 2013.
- Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, July 2010.
- Dyer, Chris, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of IBM Model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia, June 2013. Association for Computational Linguistics.
- Gao, Qin and Stephan Vogel. Parallel implementations of word alignment tool. In *Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 49–57, Columbus, Ohio, June 2008. Association for Computational Linguistics.
- Koehn, Philipp. An experimental management system. *Prague Bulletin of Mathematical Linguistics*, 94:87–96, September 2010.
- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics: Demonstration Session*, Prague, Czech Republic, June 2007.
- Li, Zhifei, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, March 2009. Association for Computational Linguistics.
- Och, Franz Josef and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, March 2003.

Address for correspondence:

Ulrich Germann
ugermann@inf.ed.ac.uk
School of Informatics
University of Edinburgh
10 Crichton Street
Edinburgh, EH8 9AB, United Kingdom



The Prague Bulletin of Mathematical Linguistics

NUMBER 100 OCTOBER 2013 19-30

QuEst — Design, Implementation and Extensions of a Framework for Machine Translation Quality Estimation

Kashif Shah^a, Eleftherios Avramidis^b, Ergun Biçici^c, Lucia Specia^a

^a University of Sheffield

^b German Research Center for Artificial Intelligence

^c Centre for Next Generation Localization, Dublin City University

Abstract

In this paper we present QuEst, an open source framework for machine translation quality estimation. The framework includes a feature extraction component and a machine learning component. We describe the architecture of the system and its use, focusing on the feature extraction component and on how to add new feature extractors. We also include experiments with features and learning algorithms available in the framework using the dataset of the WMT13 Quality Estimation shared task.

1. Introduction

Quality Estimation (QE) is aimed at predicting a quality score for a machine translated segment, in our case, a sentence. The general approach is to extract a number of features from source and target sentences, and possibly external resources and information from the Machine Translation (MT) system for a dataset labelled for quality, and use standard machine learning algorithms to build a model that can be applied to any number of unseen translations. Given its independence from reference translations, QE has a number of applications, for example filtering out low quality translations from human post-editing.

Most of current research focuses on designing feature extractors to capture different aspects of quality that are relevant to a given task or application. While simple features such as counts of tokens and language model scores can be easily extracted, feature engineering for more advanced information can be very labour-intensive. Dif-

ferent language pairs or optimisation against specific quality scores (e.g., post-editing time versus translation adequacy) can benefit from different feature sets.

QuEst is a framework for quality estimation that provides a wide range of feature extractors from source and translation texts and external resources and tools (Section 2). These range from simple, language-independent features, to advanced, linguistically motivated features. They include features that rely on information from the MT system that generated the translations, and features that are oblivious to the way translations were produced, and also features that only consider the source and/or target sides of the dataset (Section 2.1). QuEst also incorporates wrappers for a well-known machine learning toolkit, `scikit-learn`¹ and for additional algorithms (Section 2.2).

This paper is aimed at both users interested in experimenting with existing features and algorithms and developers interested in extending the framework to incorporate new features (Section 3). For the former, QuEst provides a practical platform for quality estimation, freeing researchers from feature engineering, and facilitating work on the learning aspect of the problem, and on ways of using quality predictions in novel extrinsic tasks, such as self-training of statistical machine translation systems. For the latter, QuEst provides the infrastructure and the basis for the creation of new features, which may also reuse resources or pre-processing techniques already available in the framework, such as syntactic parsers, and which can be quickly benchmarked against existing features.

2. Overview of the QuEst Framework

QuEst consists of two main modules: a feature extraction module and a machine learning module. It is a collaborative project, with contributions from a number of researchers.² The first module provides a number of feature extractors, including the most commonly used features in the literature and by systems submitted to the WMT12–13 shared tasks on QE (Callison-Burch et al., 2012; Bojar et al., 2013). It is implemented in Java and provides abstract classes for features, resources and pre-processing steps so that extractors for new features can be easily added.

The basic functioning of the feature extraction module requires a pair of raw text files with the source and translation sentences aligned at the sentence-level. Additional resources such as the source MT training corpus and language models of source and target languages are necessary for certain features. Configuration files are used to indicate the resources available and a list of features that should be extracted. It produces a CSV file with all feature values.

The machine learning module provides scripts connecting the feature file(s) with the `scikit-learn` toolkit. It also uses GPy, a Python toolkit for Gaussian Processes regression, which showed good performance in previous work (Shah et al., 2013).

¹<http://scikit-learn.org/>

²See <http://www.quest.dcs.shef.ac.uk/> for a list of collaborators.

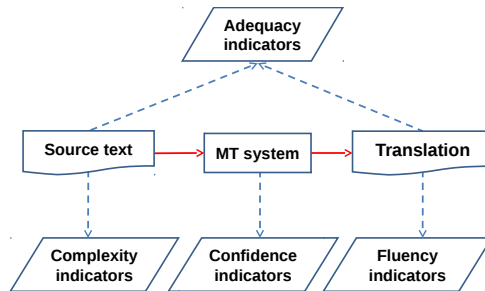


Figure 1: Families of features in QuEst.

2.1. Feature Sets

In Figure 1 we show the families of features that can be extracted in QuEst. Although the text unit for which features are extracted can be of any length, most features are more suitable for sentences. Therefore, a “segment” here denotes a sentence. Most of these features have been designed with Statistical MT (SMT) systems in mind, although many do not explore any internal information from the actual SMT system. Further work needs to be done to test these features for rule-based and other types of MT systems, and to design features that might be more appropriate for those.

From the source segments QuEst can extract features that attempt to quantify the **complexity** or **translatability** of those segments, or how unexpected they are given what is known to the MT system. From the comparison between the source and target segments, QuEst can extract **adequacy** features, which attempt to measure whether the structure and meaning of the source are preserved in the translation. Information from the SMT system used to produce the translations can provide an indication of the **confidence** of the MT system in the translations. They are called “glass-box” features (GB) to distinguish them from MT system-independent, “black-box” features (BB). To extract these features, QuEst assumes the output of Moses-like SMT systems, taking into account word- and phrase-alignment information, a dump of the decoder’s standard output (search graph information), global model score and feature values, n-best lists, etc. For other SMT systems, it can also take an XML file with relevant information. From the translated segments QuEst can extract features that attempt to measure the **fluency** of such translations.

The most recent version of the framework includes a number of previously under-explored features that can rely on only the source (or target) side of the segments and on the source (or target) side of the parallel corpus used to train the SMT system. Information retrieval (IR) features measure the closeness of the QE source sentences and their translations to the parallel training data available to predict the difficulty of translating each sentence. These have been shown to work very well in recent work

(Biçici et al., 2013; Biçici, 2013). We use Lucene³ to index the parallel training corpora and obtain a retrieval similarity score based on tf-idf. For each source sentence and its translation, we retrieve top 5 distinct training instances and calculate the following features:

- IR score for each training instance retrieved for the source sentence or its translation
- BLEU (Papineni et al., 2002) and F_1 (Biçici, 2011) scores over source or target sentences
- LIX readability score⁴ for source and target sentences
- The average number of characters in source and target words and their ratios.

In Section 4 we provide experiments with these new features.

The complete list of features available is given as part of QuEst’s documentation. At the current stage, the number of BB features varies from 80 to 143 depending on the language pair, while GB features go from 39 to 48 depending on the SMT system.

2.2. Machine Learning

QuEst provides a command-line interface module for the `scikit-learn` library implemented in Python. This module is completely independent from the feature extraction code. It reads the extracted feature sets to build and test QE models. The dependencies are the `scikit-learn` library and all its dependencies (such as NumPy and SciPy). The module can be configured to run different regression and classification algorithms, feature selection methods and grid search for hyper-parameter optimisation.

The pipeline with feature selection and hyper-parameter optimisation can be set using a configuration file. Currently, the module has an interface for Support Vector Regression (SVR), Support Vector Classification, and Lasso learning algorithms. They can be used in conjunction with the feature selection algorithms (Randomised Lasso and Randomised decision trees) and the grid search implementation of `scikit-learn` to fit an optimal model of a given dataset.

Additionally, QuEst includes Gaussian Process (GP) regression (Rasmussen and Williams, 2006) using the GPy toolkit.⁵ GPs are an advanced machine learning framework incorporating Bayesian non-parametrics and kernel machines, and are widely regarded as state of the art for regression. Empirically we found its performance to be similar or superior to that of SVR for most datasets. In contrast to SVR, inference in GP regression can be expressed analytically and the model hyper-parameters optimised using gradient ascent, thus avoiding the need for costly grid search. This also makes the method very suitable for feature selection.

³lucene.apache.org

⁴<http://en.wikipedia.org/wiki/LIX>

⁵<https://github.com/SheffieldML/GPy>

3. Design and Implementation

3.1. Source Code

We made available three versions of the code, all available from <http://www.quest.dcs.shef.ac.uk>:

- An installation script that will download the stable version of the source code, a built up version (jar), and all necessary pre-processing resources/tools (parsers, etc.).
- A stable version of the above source code only (no linguistic processors).
- A vanilla version of the source code which is easier to run (and re-build), as it relies on fewer pre-processing resources/tools. Toy resources for en-es are also included in this version. It only extracts up to 50 features.

In addition, the latest development version of the code can be accessed on GitHub.⁶

3.2. Setting Up

Once downloaded, the folder with the code contains all files required for running or building the application. It contains the following folders and resources:

- `src`: java source files
- `lib`: jar files, including the external jars required by QuEst
- `dist`: javadoc documentation
- `lang-resources`: example of language resources required to extract features
- `config`: configuration files
- `input`: example of input training files (source and target sentences, plus quality labels)
- `output`: example of extracted feature values

3.3. The Feature Extractor

The class that performs feature extraction is `shef.mt.FeatureExtractor`. It handles the extraction of glass-box and/or black-box features from a pair of source-target input files and a set of additional resources specified as input parameters. Whilst the command line parameters relate to the current set of input files, `FeatureExtractor` also relies on a set of project-specific parameters, such as the location of resources. These are defined in a configuration file in which resources are listed as pairs of key=value entries. By default, if no configuration file is specified in the input, the application will search for a default `config.properties` file in the current working folder (i.e., the folder where the application is launched from). This default file is provided with the distribution.

Another input parameter required is the XML feature configuration file, which gives the identifiers of the features that should be extracted by the system. Unless

⁶<https://github.com/lspcia/quest>

a feature is present in this feature configuration file it will not be extracted by the system. Examples of such files for all features, black-box, glass-box, and a subset of 17 “baseline” features are provided with the distribution.

3.4. Running the Feature Extractor

The following command triggers the features extractor:

```
FeatureExtractor -input <source file> <target file> -lang
<source language> <target language> -config <configuration file>
-mode [gb|bb|all] -gb [list of GB resources]
```

where the arguments are:

- `-input <source file> <target file>` (required): the input source and target text files with sentences to extract features from
- `-lang <source language> <target language>`: source and target languages of the files above
- `-config <configuration file>`: file with the paths to the input/output, XML-feature files, tools/scripts and language resources
- `-mode <gb|bb|all>`: a choice between glass-box, black-box or both types of features
- `-gb [list of files]`: input files required for computing the glass-box features. The options depend on the MT system used. For Moses, three files are required: a file with the n-best list for each target sentence, a file with a verbose output of the decoder (for phrase segmentation, model scores, etc.), and a file with search graph information.

3.5. Packages and Classes

Here we list the important packages and classes. We refer the reader to QuEst documentation for a comprehensive list of modules.

- `shef.mt.enes`: This package contains the main feature extractor classes.
- `shef.mt.features.impl.bb`: This package contains the implementations of black-box features.
- `shef.mt.features.impl.gb`: This package contains the implementations of glass-box features.
- `shef.mt.features.util`: This package contains various utilities to handle information in a sentence and/or phrase.
- `shef.mt.tools`: This package contains wrappers for various pre-processing tools and Processor classes for interpreting the output of the tools.
- `shef.mt.tools.stf`: This package contains classes that provide access to the Stanford parser output.
- `shef.mt.util`: This package contains a set of utility classes that are used throughout the project, as well as some independent scripts used for various data preparation tasks.

- `shef.mt.xmlwrap`: This package contains XML wrappers to process the output of SMT systems for glass-box features.

The most important classes are as follows:

- `FeatureExtractor`: `FeatureExtractor` extracts glass-box and/or black-box features from a pair of source-target input files and a set of additional resources specified as input parameters.
- `Feature`: `Feature` is an abstract class which models a feature. Typically, a `Feature` consist of a value, a procedure for calculating the value and a set of dependencies, i.e., resources that need to be available in order to be able to compute the feature value.
- `FeatureXXXX`: These classes extend `Feature` and to provide their own method for computing a specific feature.
- `Sentence`: Models a sentence as a span of text containing multiple types of information produced by pre-processing tools, and direct access to the sentence tokens, n-grams, phrases. It also allows any tool to add information related to the sentence via the `setValue()` method.
- `MTOutputProcessor`: Receives as input an XML file containing sentences and lists of translation with various attributes and reads it into `Sentence` objects.
- `ResourceProcessor`: Abstract class that is the basis for all classes that process output files from pre-processing tools.
- `Pipeline`: Abstract class that sets the basis for handling the registration of the existing `ResourceProcessors` and defines their order.
- `ResourceManager`: This class contains information about resources for a particular feature.
- `LanguageModel`: `LanguageModel` stores information about the content of a language model file. It provides access to information such as the frequency of n-grams, and the cut-off points for various n-gram frequencies necessary for certain features.
- `Tokenizer`: A wrapper around the Moses tokenizer.

3.6. Developer's Guide

A hierarchy of a few of the most important classes is shown in Figure 2. There are two principles that underpin the design choice:

- pre-processing must be separated from the computation of features, and
- feature implementation must be modular in the sense that one is able to add features without having to modify other parts of the code.

A typical application will contain a set of tools or resources (for pre-processing), with associated classes for processing the output of these tools. A `Resource` is usually a wrapper around an external process (such as a part-of-speech tagger or parser), but it can also be a brand new fully implemented pre-processing tool. The only requirement for a tool is to extend the abstract class `shef.mt.tools.Resource`. The implementation of a tool/resource wrapper depends on the specific requirements of that

particular tool and on the developer's preferences. Typically, it will take as input a file and a path to the external process it needs to run, as well as any additional parameters the external process requires, it will call the external process, capture its output and write it to a file.

The interpretation of the tool's output is delegated to a subclass of `shef.mt.tools.ResourceProcessor` associated with that particular Resource. A `ResourceProcessor` typically:

- Contains a function that initialises the associated Resource. As each Resource may require a different set of parameters upon initialisation, `ResourceProcessor` handles this by passing the necessary parameters from the configuration file to the respective function of the Resource.
- Registers itself with the `ResourceManager` in order to signal the fact that it has successfully managed to initialise itself and it can pass information to be used by features. This registration should be done by calling `ResourceManager.registerResource(String resourceName)`. `resourceName` is an arbitrary string, unique among all other Resources. If a feature requires this particular Resource for its computation, it needs to specify it as a requirement (see Section 3.7).
- Reads in the output of a Resource sentence by sentence, retrieves some information related to that sentence and stores it in a Sentence object. The processing of a sentence is done in the `processNextSentence(Sentence sentence)` function which all `ResourceProcessor`-derived classes must implement. The information it retrieves depends on the requirements of the application. For example, `shef.mt.tools.POSProcessor`, which analyses the output of the `TreeTagger`, retrieves the number of nouns, verbs, pronouns and content words, since these are required by certain currently implemented features, but it can be easily extended to retrieve, for example, adjectives, or full lists of nouns instead of counts.

A Sentence is an intermediate object that is, on the one hand, used by `ResourceProcessor` to store information and, on the other hand, by `Feature` to access this information. The implementation of the Sentence class already contains access methods to some of the most commonly used sentence features, such as the text it spans, its tokens, its n-grams, its phrases and its n-best translations (for glass-box features). For a full list of fields and methods, see the associated javadoc. Any other sentence information is stored in a `HashMap` with keys of type `String` and values of generic type `Object`. A pre-processing tool can store any value in the `HashMap` by calling `setValue(String key, Object value)` on the currently processed Sentence object. This allows tools to store both simple values (integer, float) as well as more complex ones (for example, the `ResourceProcessor`).

A Pipeline defines the order in which processors will be initialised and run. They are defined in the `shef.mt.pipelines` package. They allow more flexibility for the execution of pre-processors, when there are dependencies between each other. At the moment `QuEst` offers a default pipeline which contains the tools required for the "vanilla" version of the code and new `FeatureExtractors` have to register there. A

more convenient solution would be a dynamic pipeline which automatically identifies the processors required by the enabled features and then initialises and runs only them. This functionality is currently under development in QuEst.

3.7. Adding a New Feature

In order to add a new feature, one has to implement a class that extends `shef.mt.features.impl.Feature`. A Feature will typically have an index and a description which should be set in the constructor. The description is optional, whilst the index is used in selecting and ordering the features at runtime, therefore it should be set. The only function a new Feature class has to implement is `run(Sentence source, Sentence target)`. This will perform some computation over the source and/or target sentence and set the return value of the feature by calling `setValue(float value)`. If the computation of the feature value relies on some pre-processing tools or resources, the constructor can add these resources or tools in order to ensure that the feature will not run if the required files are not present. This is done by a call to `addResource(String resource_name)`, where `resource_name` has to match the name of the resource registered by the particular tool this feature depends on.

4. Benchmarking

In this section we briefly benchmark QuEst using the dataset of the main WMT13 shared task on QE (subtask 1.1) using all our features, and in particular the new source-based and IR features. The dataset contains English-Spanish sentence translations produced by an SMT system and judged for post-editing effort in [0,1] using TERp,⁷ computed against a human post-edited version of the translations (i.e. HTER). 2,254 sentences were used for training, while 500 were used for testing.

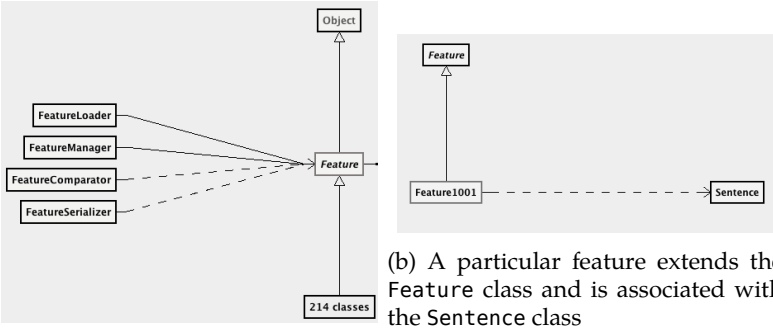
As learning algorithm we use SVR with radial basis function (RBF) kernel, which has been shown to perform very well in this task (Callison-Burch et al., 2012). The optimisation of parameters is done with grid search based on pre-set ranges of values as given in the code distribution.

For feature selection, we use Gaussian Processes. Feature selection with Gaussian Processes is done by fitting per-feature RBF widths. The RBF width denotes the importance of a feature, the narrower the RBF the more important a change in the feature value is to the model prediction. To avoid the need of a development set to optimise the number of selected features, we select the 17 top-ranked features (as in our baseline system) and then train a model with these features.

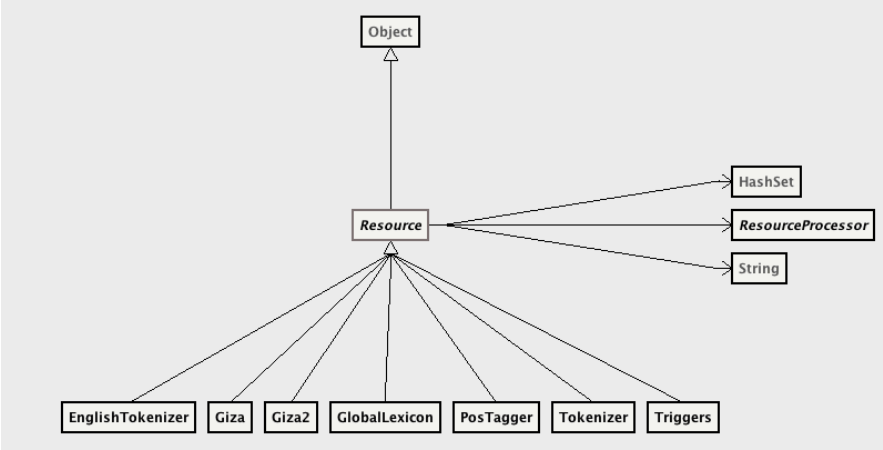
For given dataset we build the following systems with different feature sets:

- **BL:** 17 baseline features that have been shown to perform well across languages in previous work and were used as a baseline in the WMT12 QE task

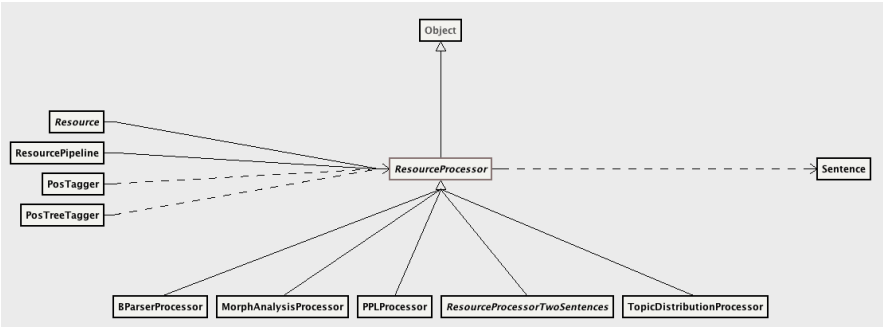
⁷<http://www.umiacs.umd.edu/~snover/terp/>



(a) The Feature class



(c) An abstract Resource class acts as a wrapper for external processes



(d) ResourceProcessor reads the output of a tool and stores it in a Sentence object

Figure 2: Class hierarchy for most important classes.

- **AF**: All features available from the latest stable version of QuEst, either black-box (BB) or glass-box (GB)
- **IR**: IR-related features recently integrated into QuEst (Section 2.1)
- **AF+IR**: All features available as above, plus recently added IR-related features
- **FS**: Feature selection for automatic ranking and selection of top features from all of the above with Gaussian Processes.

Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are used to evaluate the models. The error scores for all feature sets are reported in Table 1. Bold-faced figures are significantly better than all others (paired t-test with $p \leq 0.05$).

Feature type	System	#feats.	MAE	RMSE
BB	Baseline	17	14.32	18.02
	IR	35	14.57	18.29
	AF	108	14.07	18.13
	AF+IR	143	13.52	17.74
	FS	17	12.61	15.84
GB	AF	48	17.03	20.13
	FS	17	16.57	19.14
BB+GB	AF	191	14.03	19.03
	FS	17	12.51	15.64

Table 1: Results with various feature sets.

Adding more BB features (systems **AF**) improves the results in most cases as compared to the baseline systems **BL**, however, in some cases the improvements are not significant. This behaviour is to be expected as adding more features may bring more relevant information, but at the same time it makes the representation more sparse and the learning prone to overfitting. Feature selection was limited to selecting the top 17 features for comparison with our baseline feature set. It is interesting to note that system **FS** outperformed the other systems in spite of using fewer features.

GB features on their own perform worse than BB features but the combination of GB and BB followed by feature selection resulted in lower errors than BB features only, showing that the two features sets can be complementary, although in most cases BB features suffice. These are in line with the results reported in (Specia et al., 2013; Shah et al., 2013). A system submitted to the WMT13 QE shared task using QuEst with similar settings was the top performing submission for Task 1.1 (Beck et al., 2013).

5. Remarks

The source code for the framework, the datasets and extra resources can be downloaded from <http://www.quest.dcs.shef.ac.uk/>. The project is also set to receive contribution from interested researchers using a GitHub repository. The license for

the Java code, Python and shell scripts is BSD, a permissive license with no restrictions on the use or extensions of the software for any purposes, including commercial. For pre-existing code and resources, e.g., `scikit-learn`, `GPY` and `Berkeley parser`, their licenses apply, but features relying on these resources can be easily discarded if necessary.

Acknowledgements

This work was supported by the QuEst (EU FP7 PASCAL2 NoE, Harvest program) and QTLaunchPad (EU FP7 CSA No. 296347) projects. We would like to thank our many contributors, especially José G. C. Souza for the integration with `scikit-learn`, and Lukas Poustka for his work on the refactoring of some of the code.

Bibliography

- Beck, Daniel, Kashif Shah, Trevor Cohn, and Lucia Specia. SHEF-Lite: When less is more for translation quality estimation. In *Proceedings of WMT13*, pages 337–342, Sofia, 2013.
- Biçici, E. *The Regression Model of Machine Translation*. PhD thesis, Koç University, 2011.
- Biçici, E. Referential translation machines for quality estimation. In *Proceedings of WMT13*, pages 341–349, Sofia, 2013.
- Biçici, E., D. Groves, and J. van Genabith. Predicting sentence translation quality using extrinsic and language independent features. *Machine Translation*, 2013.
- Bojar, O., C. Buck, C. Callison-Burch, C. Federmann, B. Haddow, P. Koehn, C. Monz, M. Post, R. Soricut, and L. Specia. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of WMT13*, pages 1–44, Sofia, 2013.
- Callison-Burch, C., P. Koehn, C. Monz, M. Post, R. Soricut, and L. Specia. Findings of the 2012 Workshop on Statistical Machine Translation. In *Proceedings of WMT12*, pages 10–51, Montréal, 2012.
- Papineni, K., S. Roukos, T. Ward, and W. Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th ACL*, pages 311–318, Philadelphia, 2002.
- Rasmussen, C.E. and C.K.I. Williams. *Gaussian processes for machine learning*, volume 1. MIT Press, Cambridge, 2006.
- Shah, K., T. Cohn, and L. Specia. An investigation on the effectiveness of features for translation quality estimation. In *Proceedings of MT Summit XIV*, Nice, 2013.
- Specia, L., K. Shah, J. G. C. Souza, and T. Cohn. QuEst – a translation quality estimation framework. In *Proceedings of the 51st ACL: System Demonstrations*, pages 79–84, Sofia, 2013.

Address for correspondence:

Kashif Shah
 Kashif.Shah@sheffield.ac.uk
 Department of Computer Science
 University of Sheffield
 Regent Court, 211 Portobello, Sheffield, S1 4DP UK



The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013 31-40

MTMonkey: A Scalable Infrastructure for a Machine Translation Web Service

Aleš Tamchyna, Ondřej Dušek, Rudolf Rosa, Pavel Pecina

Charles University in Prague, Faculty of Mathematics and Physics, Institute of Formal and Applied Linguistics

Abstract

We present a web service which handles and distributes JSON-encoded HTTP requests for machine translation (MT) among multiple machines running an MT system, including text pre- and post-processing. It is currently used to provide MT between several languages for cross-lingual information retrieval in the EU FP7 Khresmoi project. The software consists of an application server and remote workers which handle text processing and communicate translation requests to MT systems. The communication between the application server and the workers is based on the XML-RPC protocol. We present the overall design of the software and test results which document speed and scalability of our solution. Our software is licensed under the Apache 2.0 licence and is available for download from the Lindat-Clarín repository and Github.

1. Introduction

In this paper, we describe an infrastructure for a scalable machine translation web service capable of providing MT services among multiple languages to remote clients posting JSON-encoded requests.

The infrastructure was originally developed as a component of the EU FP7 Khresmoi project, a multilingual multimodal search and access system for biomedical information and documents (Aswani et al., 2012), to provide MT services for real-time translation of user queries and retrieved document summaries. The service is used with three language pairs (Czech–English, French–English, and German–English) in both directions within the Khresmoi project, but the system is designed to be language-independent and capable of serving multiple translation directions.

For Khresmoi to run smoothly, the translation system must be able to quickly and reliably react to translation requests, typically with multiple requests arriving at the same time. Since machine translation is a highly computationally demanding task, solutions as efficient as possible must be sought. The system must also contain error detection and recovery mechanisms to ensure uninterrupted operation of the service. Moreover, the solution must be naturally scalable to allow for flexible increase of computational power to reach higher performance if required by its customers' demand.

In this paper, we describe the structure of our translation system, and detail the results of several performance tests. We make the system available as free software, licensed under the Apache 2.0 licence.¹ MTMonkey 1.0 is published via the Lindat-Clarín repository,² updated code is released on GitHub and open for comments and further contributions.³

2. Pre-considerations

We build upon Moses (Koehn et al., 2007), a statistical machine translation system. Koehn (2013, Section 3.3.22) explains how to operate Moses as *Moses Server* responding to translation requests on a given port. Support for using multiple translation directions was originally available as *Using Multiple Translation Systems in the Same Server* (Koehn, 2013, p. 121), later to be replaced by more general *Alternate Weight Settings* (Koehn, 2013, p. 135), which is still under development and currently does not work with multi-threaded decoding. We therefore decided to handle different translation directions using separate stand-alone Moses Server instances.

Moses does not provide any built-in support for load balancing, which is needed to distribute the translation requests evenly among the Moses instances. We therefore explored RabbitMQ,⁴ a robust open-source messaging toolkit which can be used to implement even complex application communication scenarios. However, we concluded that for our relatively simple task where the main focus is on efficiency, its overhead is unpleasant while the benefits it brings are only moderate. We therefore decided to implement our own solution for request distribution and load balancing.

We implement our solution in Python, which was chosen due to its relatively high efficiency combined with the comfortable programming experience it offers.

There are several remote procedure call (RPC) protocols available that could be used in our system. For the public API, we use JSON-RPC,⁵ which is simple and lightweight in comparison to other RPC protocols, making it highly suitable for RPC

¹<http://www.apache.org/licenses/LICENSE-2.0>

²<http://hdl.handle.net/11858/00-097C-0000-0022-AAF5-B>

³<https://github.com/ufal/mtmonkey>

⁴<http://www.rabbitmq.com/>

⁵<http://www.jsonrpc.org/>

over the Internet (other formats could be easily added if needed). Moses Server implements XML-RPC,⁶ which is similar to JSON-RPC, although not as lightweight. We employ XML-RPC for the internal API as well, since it has a native Python implementation, which is more efficient and seamless than JSON-RPC Python libraries.

MTMonkey is in its architecture very similar to the MT Server Land system (Ferdemann and Eisele, 2010), which uses XML-RPC as a response format and focuses more on the possibility of comparing different MT systems for the same translation direction than on low-latency processing of a large number of simultaneous requests. A similar approach to ours was also taken by Arcan et al. (2013), who built a multilingual financial term translation system on top of Moses.⁷ They make their system freely available through both a web GUI and a RESTful service, using JSON as the response format. They provide lists of n-best translations and allow the users to upload their own dictionaries, which are used to override the SMT system-generated translations. The WebTranslation toolkit⁸ for translating web pages which is built into Moses also supports distributing translation requests to multiple instances of Moses servers but this solution is a proof of concept only and not designed for production environments.

3. Implementation

MTMonkey consists of an *application server* and a set of *workers*. The application server handles translation request arriving through the public API and uses the internal API to distribute them to the workers, which perform the translations. The system is able to handle multiple incoming translation requests by load balancing and queuing. Self-check mechanisms are also included. The architecture of the system is visualized in Figure 1 and described in detail in Sections 3.1–3.6.

The application server and workers are implemented in Python and are compatible with Python versions 2.6 and 2.7. The installation and support scripts are written in Bash. In addition, we provide a very simple PHP-based web client that allows for an easy interactive testing of the service and serves as an example client implementation. We tested the whole system under Ubuntu 10.04, but it should be able to operate on any Unix-like system.

3.1. Public API

The application server provides a public API based on the REST⁹ principles, accepting requests over HTTP in the JSON format as objects with the following keys:

⁶<http://www.xmlrpc.com/>

⁷<http://monnet01.sindice.net/monnet-translation/>

⁸<http://www.statmt.org/moses/?n=Moses.WebTranslation>

⁹http://en.wikipedia.org/wiki/Representational_state_transfer

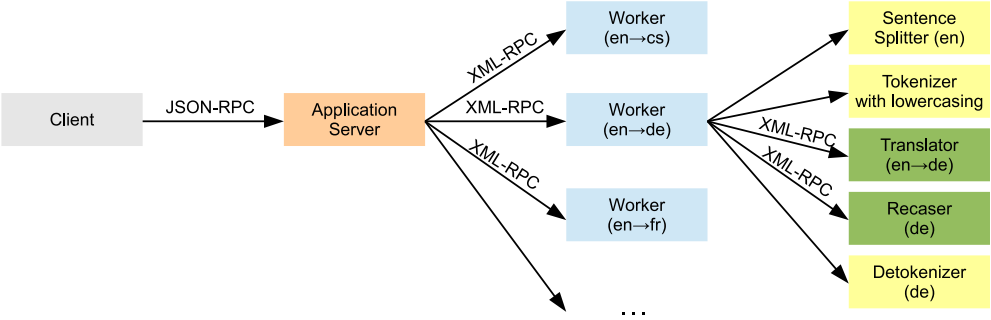


Figure 1. The overall architecture of the translation system. English-to-German translation is shown in detail.

sourceLang	the ISO 639-1 code of the source language (cs, de, en, fr);
targetLang	the ISO 639-1 code of the target language (cs, de, en, fr);
text	the text to be translated, in the UTF-8 character encoding;
detokenize	detokenize the translation (boolean);
alignmentInfo	request alignment information (boolean).

The response is a JSON object with the following keys:

errorCode	0, or error code;
translation	the translation, in the UTF-8 character encoding;
alignment-raw	alignment information (if requested by <code>alignmentInfo</code>) as a list of objects containing indexes of start- and end-tokens of corresponding source and target text chunks.

The only currently implemented advanced feature is the option to request alignment information, which can be used to determine which part of the input texts corresponds to which part of the translation. There are several other fields reserved for future use, such as `nBestSize` to request multiple translation options.¹⁰ For simplicity, we omit description of parts of the API that are unused at the moment or that are only technical.

¹⁰Due to preparation for a future implementation of the `nBestSize` option, the actual structure of the response is more complicated than described, with the actual text of the translation being wrapped in an object that itself is a member of an array of translation options.

3.2. Application Server

The application server distributes incoming translation requests to individual workers. Available workers are listed in a simple configuration file – for each worker, its IP address, port, and translation direction (source and target language) are given. Because the workers are identified by a combination of the IP address and port number, there can be multiple workers on one machine listening on different ports.

If there are multiple workers available for a given translation direction, a simple round-robin load balancing is used. No other information, such as text length or worker configuration, is taken into account. However, we found that such a simple approach is sufficient for our needs, and at the same time it is fast enough not to unnecessarily increase the response time, making the application server lightweight enough to require only moderate computational resources. If more machines support several translation directions, a set of translation requests for that direction can be distributed relatively evenly among all the respective machines. The number of workers is potentially unlimited, i.e. the only limit is the available computational power.

3.3. Internal API

The application server communicates with workers through XML-RPC. A worker implements two XML-RPC methods:

process_task	used to request a translation, returning the translated text (with additional information if requested, such as the alignment);
alive_check	tests if the worker is running.

3.4. Workers

Each worker uses one instance of Moses providing translation in one direction and another instance of Moses that performs recasing. The only configuration parameters of a worker are the ports on which the Moses servers listen. The worker communicates with the Moses servers through XML-RPC. Workers run as multi-threaded XML-RPC servers which allows for transparent and light-weight asynchronous processing and parallelism. One physical machine may house multiple instances of a worker, each using its own MT system instance, providing translation in a different direction. Only the available RAM and hard drive space are the limits on the number of running worker instances.

3.5. Text Processing Tools

The input texts have to be preprocessed before translation. We use the usual pipeline of a sentence splitter and a lowercasing tokenizer. The sentence splitter is our reimplement of the Moses sentence splitter in Python and uses the same non-breaking prefixes definition files.

Due to our system being used as a component of a complex project, the sources of incoming translation requests are varied, and the texts to be translated can appear in various tokenizations. We therefore implemented our own language-independent tokenizer, which is robust with respect to possible pre-tokenization. We achieve this by “aggressive tokenization”: splitting the text on any punctuation, including hyphens compounds and full stops in abbreviations (but keeping sequences of identical punctuation marks unsplit, as in “...”). Although such approach might hurt translation fluency, it helps prevent data sparsity. The same approach must be applied on the training data.

As a post-processing step, we use a Moses instance to perform recasing and a detokenizer, which is our reimplement of the Moses detokenizer in Python.

3.6. Fault Recovery

To ensure uninterrupted operation, worker machines may be configured to perform scheduled self-tests and automatically restart the worker application as well as Moses servers in case of an error. We provide a testing script that may be added to the machines’ *crontab*.

In addition, we run automatic external tests that are scheduled to translate a test sentence and notify the administrator of the service by e-mail in case of any error. These tests connect to the service in exactly the same way as other clients, i.e. they reflect the actual service state from the outside.

4. Evaluation

The evaluation presented in this section is focused on efficiency. We measure how fast the system is in serving various numbers of simultaneous requests.

4.1. System Configuration

We test the system using eight worker machines, each with four CPUs and 32 GB RAM. Each of the machines runs three worker instances (each for a different translation direction), i.e. there are four workers for each translation direction.

We use binarized models (for both the phrase-table and the language model) with lazy loading in Moses, which causes a slight decrease in translation speed.¹¹ However, this setup gives us more flexibility as it allows us to fit multiple instances of Moses into RAM on a single machine and begin translating almost instantly after starting the Moses servers. More details about the setup of the Moses translation system itself can be found in Pecina et al. (2012).

¹¹ The decrease in speed is noticeable even for batch translation using a single system.

4.2. Load Testing

To generate translation requests, we use two data sets, both created within the Khresmoi project. The first set consists of *sentences* from the medical domain with 16.2 words per sentence on average. The second set consists of medical search *queries* with an average length of 2.1 words per query.

In each of the tests, we run a number of clients simultaneously, either for one translation direction at a time, or for all six of them. Each of the clients sends 10 synchronous translation requests to the application server and reports the time elapsed for all of them to complete, which (divided by 10) gives the average response time. To test the scalability of our solution, we also run some of the tests with a reduced number of workers. The one-translation-direction tests were run separately for each of the six translation directions.¹² The tests were repeated 10 times with different parts of the test data.¹³ The results were then averaged and the standard deviation was computed.

The results are shown in Table 1. We average the results over all translation directions since we observed that there are only little differences in performance with respect to the translation direction (less than 15% of the average response time). We can see that when moving from one client to 10 clients, the number of parallel requests rises faster than the average time needed to complete them. This indicates that the parallelization and load balancing function properly. However, the standard deviation is relatively large, which indicates that the load balancing probably could be improved. If we multiply the number of parallel requests by 10 one more time, the average request time gets also approximately multiplied by 10, indicating that the parallelization capacity has already been reached at that point.

The scalability tests revealed that with a large number of parallel requests, doubling the number of workers reduces the response time to approximately a half. This shows that the system scales well, with a possibility to reach low response times even under high load (the minimum average response time being around 550ms for sentence translations in our setup) provided that sufficient computational power is available.

In spite of the queries being more than seven times shorter than the sentences on average, the query translation was observed to be only up to five times faster than the sentence translation under low load, and becomes consistently only about twice as fast with higher numbers of parallel requests. This indicates that the length of the input texts is not as crucial for the system performance as other parameters.

¹²The 6-translation-directions tests were not run with 100 clients per direction since we are technically unable to run 600 clients in parallel.

¹³Except for the 100-client test which uses all of the data and was therefore run only once.

Data type	Translation directions	Clients per direction	Workers per direction	Response time [ms]	
				avg	std dev
sentences	1	1	1	539	132
sentences	1	1	2	510	134
sentences	1	1	4	554	151
sentences	1	10	1	2,178	506
sentences	1	10	2	897	259
sentences	1	10	4	567	171
sentences	1	100	1	14,941	2,171
sentences	1	100	2	10,189	1,588
sentences	1	100	4	5,560	794
sentences	6	1	1	620	137
sentences	6	1	2	571	143
sentences	6	1	4	592	196
sentences	6	10	1	4,792	857
sentences	6	10	2	2,103	408
sentences	6	10	4	1,029	280
queries	1	1	4	112	29
queries	1	10	4	247	149
queries	1	100	4	2,593	526
queries	6	1	4	174	110
queries	6	10	4	545	91

Table 1. Load testing results.

5. Conclusion

We described a successful implementation of a machine translation web service that is sufficiently robust and fast enough to handle parallel translation requests in several translation directions at once and can be easily scaled to increase performance.

Our future plan is to implement worker hot-plugging for an even more flexible scalability, as currently adding or removing workers requires a restart of the application server. We also intend to add the drafted advanced features of the API, such as requesting and returning multiple translation options and their scores. We are also planning to develop a simple confidence-estimation module to assess the quality of produced translations.

We further plan to enrich the APIs with a method capable of retrieving diagnostic and statistical information, such as the list of supported translation directions, the number of workers for each translation direction, average response time or the number of requests served in the last hour. We would also like to add support for other MT decoders besides Moses.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257528 (KHRESMOI) and the project DF12P01OVV022 of the Ministry of Culture of the Czech Republic (NAKI – Amalach).

This work has been using language resources developed and/or stored and/or distributed by the LINDAT-Clarin project of the Ministry of Education of the Czech Republic (project LM2010013).

Bibliography

- Arcan, Mihael, Susan Marie Thomas, Derek De Brandt, and Paul Buitelaar. Translating the FIN-REP taxonomy using a domain-specific corpus. In *Machine Translation Summit XIV*, Nice, France, 2013.
- Aswani, Niraj, Thomas Beckers, Erich Birngruber, Célia Boyer, Andreas Burner, Jakub Bystroň, Khalid Choukri, Sarah Cruchet, Hamish Cunningham, Jan Dědek, Ljiljana Dolamic, René Donner, Sebastian Dungs, Ivan Eggel, Antonio Foncubierta-Rodríguez, Norbert Fuhr, Adam Funk, Alba García Seco de Herrera, Arnaud Gaudinat, Georgi Georgiev, Julien Gobeill, Lorraine Goeuriot, Paz Gómez, Mark Greenwood, Manfred Gschwandtner, Allan Hanbury, Jan Hajič, Jaroslava Hlaváčová, Markus Holzer, Gareth Jones, Blanca Jordan, Matthias Jordan, Klemens Kaderk, Franz Kainberger, Liadh Kelly, Sascha Kriewel, Marlene Kritz, Georg Langs, Nolan Lawson, Dimitrios Markonis, Ivan Martinez, Vassil Momtchev, Alexandre Masselot, Hélène Mazo, Henning Müller, Pavel Pecina, Konstantin Pentchev, Deyan Peychev, Natalia Pletneva, Diana Pottecher, Angus Roberts, Patrick Ruch, Matthias Samwald, Priscille Schneller, Veronika Stefanov, Miguel A. Tinte, Zdeňka Urešová, Alejandro Vargas, and Dina Vishnyakova. Khresmoi: Multimodal multilingual medical information search. In *Proceedings of the 24th International Conference of the European Federation for Medical Informatics*, 2012. URL <http://publications.hevs.ch/index.php/attachments/single/458>.
- Federmann, Christian and Andreas Eisele. MT Server Land: An open-source MT architecture. *Prague Bulletin of Mathematical Linguistics*, 94:57–66, 2010.
- Koehn, Philipp. Moses, statistical machine translation system, user manual and code guide, July 2013. URL <http://www.statmt.org/moses/manual/manual.pdf>.
- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P/P07/P07-2045>.

Pecina, Pavel, Jakub Bystroň, Jan Hajič, Jaroslava Hlaváčová, and Zdeňka Uřešová. Deliverable 4.3: Report on results of the WP4 first evaluation phase. Public deliverable, Khresmoi project, 2012. URL <http://www.khresmoi.eu/assets/Deliverables/WP4/Khresmoid43.pdf>.

Address for correspondence:

Aleš Tamchyna
tamchyna@ufal.mff.cuni.cz
Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics,
Charles University in Prague
Malostranské náměstí 25
118 00 Praha 1, Czech Republic



The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013 41-50

**DIMwid — Decoder Inspection for Moses
(using Widgets)**

Robin Kurtz, Nina Seemann, Fabienne Braune, Andreas Maletti

University of Stuttgart, Institute for Natural Language Processing
Pfaffenwaldring 5b, D-70569 Stuttgart, Germany

Abstract

The development of accurate machine translation systems requires detailed analyses of the recurring translation mistakes. However, the manual inspection of the decoder log files is a daunting task because of their sheer size and their uncomfortable format, in which the relevant data is widely spread. For all major platforms, DIMwid offers a graphical user interface that allows the quick inspection of the decoder stacks or chart cells for a given span in a uniform way. Currently, DIMwid can process the decoder log files of the phrase-based stack decoder and the syntax-based chart decoder inside the Moses framework.

1. Introduction

Statistical machine translation is the research area that concerns itself with the development of automatic translation systems for natural language text using statistical processes. The last decade saw significant progress in the translation quality due to improved models and the availability of huge parallel text corpora. These corpora are used to automatically obtain translation rules, which are then weighted according to their usefulness. In this way, the translation model is obtained. In the decoding step, this model is applied to an input sentence to produce a translation of it. Modern statistical machine translation systems, such as GOOGLE TRANSLATE, are widely used nowadays and offer reasonable access to foreign languages. Naturally, the translations produced by those automatic systems are not perfect yet, but the gist can often be understood.

Frameworks such as MOSES (Koehn et al., 2007) allow the fast and simple development of state-of-the-art statistical machine translation systems. The natural first step towards improving such a system consists of a detailed analysis of the recurring errors occurring in the baseline system. Once the problematic translations are identified, we would like to investigate how the translation was obtained from the rules. Besides the rules that were used in the translation we would also like to identify the competing rules and check whether a more suitable translation is possible in principle. Finally, we need to find out why the problematic translation was preferred over better translations.

These analysis steps are fully supported by the MOSES framework, but require a manual inspection of the trace log of the decoder. The trace contains all the relevant information in plain text and can easily be used by experienced MOSES developers, who know what to look out for. For novices the trace is not accessible at all because of its cryptic format and its sheer size. Our open-source tool *DIMwid* addresses this problem by providing a graphical user interface that displays the trace in a more user-friendly manner. A chart displays all translation items grouped according to the source span that they cover. *DIMwid* can display all standard traces of the MOSES decoders in this manner and we also added a new trace allowing us to better identify used rule in the syntax-based chart decoder.

1.1. Design Choices

DIMwid should allow inexperienced MOSES users to identify problematic translations, inspect the decoder trace to find the problematic rules and even find the competing rules that would potentially enable a better translation. To this end, *DIMwid* displays the trace output, which consists of the decoder stack items or chart items, in a uniform chart-based format. Each item is associated to the source span that it covers. This deviates from the grouping used in the standard stack decoder, but makes the analysis simpler. In fact, our goal was to make the tool simple enough to be useful for instructors in class. Naturally, each item typically comes with a variety of additional information (such as partial scores) and we display this information in the detailed view. Using this information and knowledge about the general decoding process, we can reconstruct how the decoder processed the sentence and which alternatives were considered. To streamline the inspection process, *DIMwid* can load the trace of multiple input sentences and allows opening several detailed views of items, which allow an easy comparison.

Besides the core functionality, we aimed to make the tool open-source and available on all major operating systems, which we achieved using the graphical framework Qt and the programming language PYTHON, which is one of the most commonly used programming languages. In addition, PYTHON source-code is easily readable and thus a popular choice for open-source projects. Finally, PYTHON also supports our last goal, which was to build the architecture such that extensions and adjustments can

Language/Framework	Minimal Version		
	LINUX	MACOS	WINDOWS
PYTHON	2.7.3	2.7.3	2.7.5
QT	4.8.4	4.8.2	4.0
PYQT	3.18.1	4.9.4	4.10.1

Table 1. List of required packages together with their minimal tested versions.

easily be made to support future decoders and to satisfy the specific analysis requirements of users.

1.2. Related Work

The statistical machine translation framework JOSHUA (Li et al., 2009) already offers a graphical tool (Weese and Callison-Burch, 2009) for analyzing the translations. JOSHUA uses a syntax-based translation model and the visualization shows the hypergraph of the n -best candidate translations obtained during decoding. MOSES is often used for phrase-based machine translation and we decided to use a CYK-parsing like chart, which scales better and should be similarly instructive to non-experts.

2. Installation

DIMwid requires the packages PYTHON, QT, and PYQT. The minimal required (and tested) versions of these packages are listed in Table 1. On LINUX-based systems (such as UBUNTU, FEDORA, OPENSUSE), these packages can normally be installed via the operating system’s package manager. The installation under WINDOWS requires the manual download and the execution of the package installers, but the installation is straightforward. We note that recent WINDOWS packages for PyQt already contain the QT framework, so WINDOWS users only need to install PYTHON and PyQt. The situation is similar for MacOS users. They only need to install a recent PYTHON and a binary package for PyQt. We recommend PyQTX, whose complete installation includes a compatible version of QT.

DIMwid itself is available on GITHUB at

<https://github.com/RobinQrtz/DIMwid>

under the MIT license, which allows *DIMwid* and its source code to be used freely for all purposes. *DIMwid* does not need to be installed or prepared since it only consists of three PYTHON source files. However, to successfully use it we need the trace files of the decoders inside the MOSES machine translation framework.

3. Usage

3.1. Obtaining the Input

DIMwid can process all major output formats that are produced by the decoders inside the MOSES framework and 2 new custom formats:

- the full decoder trace of the chart-based decoder (-TALL option) and
- the full decoder trace of the chart-based decoder of the multi bottom-up tree transducer extension (Braune et al., 2013).

A recent version of the (master) MOSES framework is required for the -TALL option and the MBOTDECODER branch of MOSES is needed for the second custom format. We refer the reader to the technical documentation for the use of *DIMwid* in conjunction with the MBOTDECODER, but next we recall how to obtain the required trace files for the standard decoders.

3.1.1. Standard Moses trace

DIMwid supports the standard MOSES trace outputs for both the phrase-based stack decoder (Koehn et al., 2003), via the “Phrase” format, and the syntax-based chart decoder (Chiang, 2007; Hoang et al., 2009) for hierarchical and tree-based models, via the “Syntax” format. By default, these traces are obtained by calling the decoders with the -t (phrase-based) and -T (syntax-based) flags as in:

```
cat input | moses -f moses.ini -t > out
cat input | moses_chart -f moses.ini -T trace.log > out
```

These traces only contain information about the best translation for each input sentence and are thus reasonably small. They allow us to reconstruct how the reported translations were obtained from the rules and the input sentences.

3.1.2. Full stack and chart trace

Sometimes the information about the reported best translations is not sufficient for a successful analysis. The full stack or chart trace records much more information about the decoding process. It contains all items that are present in the stacks (used in phrase-based decoding) or the chart cells (used in syntax-based decoding). Consequently, it allows us to effectively inspect which hypotheses were considered by the decoder and to investigate the competing hypotheses. Not surprisingly, those traces tend to be huge.

For phrase-based decoding, the stack trace is obtained by enabling level 3 verbosity and logging the error output. This is typically achieved by:

```
cat input | moses -f moses.ini -v 3 2> trace.log > out
```

Unfortunately, running MOSES with multiple worker threads (option -threads) ruins the desired output, since the outputs are not synchronized. Consequently, the

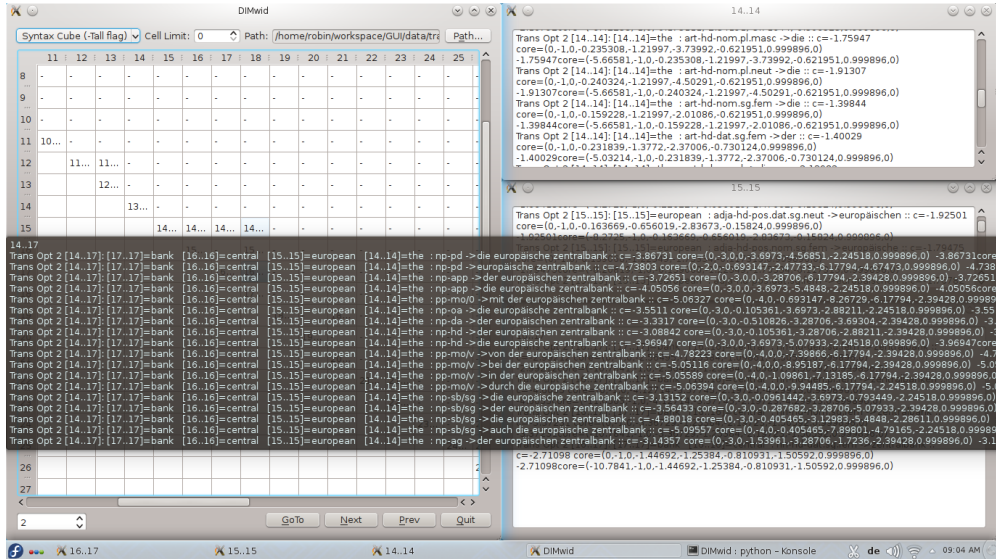


Figure 1. DIMwid presenting the chart and several details of the syntax-based decoder (display: KDE).

threads-option should not be present (neither in the call nor in the initialization file `moses.ini`) or explicitly set to “-threads 1”. The obtained trace contains all translation options and recombination and stack information organized in a standard parse chart based on the covered spans. Since the trace is typically huge, it may take *DIMwid* a while to load it.

Alternatively, MOSES also offers a flag named `-output-search-graph`, which outputs the entire search-space for the translation. This flag works for the phrase-based stack decoder and the syntax-based chart decoder. Since the output formats are different, the user needs to select the correct input format:

- “Phrase Stack (search-graph)” for the stack decoder or
- “Syntax Cube (search-graph)” for the chart decoder

in *DIMwid* when importing these traces.

For the syntax-based chart decoder we are also interested in the source-side of the used rules. This information is not provided in any of the existing formats, so we added a new output trace to MOSES, which delivers also this information. The new flag is called `-Tall` and is used in the same way as the `-T` flag. A typical call (with some advanced options) might be:

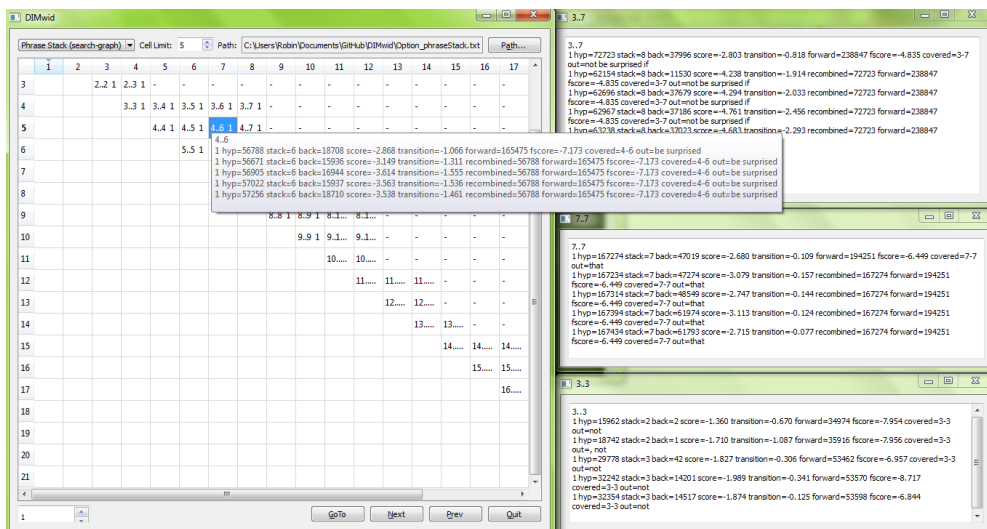


Figure 2. Display of the search-graph of a phrase-based decoder (display: Windows).

```
cat input | moses_chart -f moses.ini
    -include-lhs-in-search-graph -n-best-list listfile 100
    -T trace.log -Tall traceAll.log > out
```

The `-Tall` flag triggers the desired output, while the other flags `-n-best-list` and `-include-lhs-in-search-graph` produce more translation options and include the left-hand-sides of the rules. However, the output of the chart trace triggered by the `-T` and `-Tall` flags can be surprising. Due to Moses' internal decoding process the source-side nonterminal gets projected onto the corresponding nonterminal of the target-side. Thus, the reported source-side nonterminal might be different from the actual source-side nonterminal in the used rule, which should be considered when identifying the responsible rule. Nevertheless, the `-Tall` trace offers important information and prints — in contrast to the search graph — the correct terminals of the source-side.

3.2. Graphical User Interface

DIMwid is started by running `DIMwid.py`. The resulting main window behaves like any other window of the operating system and can therefore be maximized, minimized, moved around, etc. Keyboard commands are triggered by holding the keyboard's "Alt" key plus the underlined letter of the button. Next, we show the general steps needed to display a decoder trace:

1. First, we select the correct trace format by clicking on the “Format” button, which will open a drop-down menu containing buttons labelled with the supported formats. Once a format is selected, it will be shown on the button (instead of “Format”).
2. Optionally, we may want to limit the number of items per chart cell using the text-field next to the “Cell Limit” label. Unless the complete stack or chart information is essential, a reasonable bound on the displayed items is generally recommended because *DIMwid* tends to run slow when huge numbers of items need to be displayed.
3. Next, we select the trace file by clicking on the “Path” button. The standard file selection dialog of the operating system will open and will allow the selection of the desired file. Currently, *DIMwid* does not auto-detect the trace format, so if the file format does not correspond to the selection in Step 1, then an error message is displayed.
4. Once *DIMwid* finishes loading the trace file, the chart display is available for each input sentence. Automatically, the chart for the first input sentence, numbered 0, is displayed. Other input sentences can be selected directly by entering their number in the input box next to the “GoTo” button and pressing that button. Alternatively, the “Prev” and “Next” buttons can be used to cycle through the input sentences.
5. The content of each chart cell is explained in Section 3.3. It can be shown in two ways: (a) A tool-tip window preview of the chart cell content is displayed when moving the mouse cursor over the chart cell. (b) A double-click on the chart cell opens a new window displaying the full cell content. Multiple windows allow the comfortable comparison of the contents of several cells.

3.3. Chart Display

Once the trace is successfully loaded, a quadratic table is displayed with a blank lower left triangle (see Figure 2). It has as many rows and columns as there are words in the selected source sentence. Each chart cell corresponds to a span in the input sentence. Its row marks the beginning of the span and its column marks the end of the span. Correspondingly, the entry (m, n) contains the translations of the span $[m + 1, n + 1]$ of the source sentence, which starts at the $(m + 1)^{\text{th}}$ word and ends at the $(n + 1)^{\text{th}}$ word. The diagonal (where $m = n$) contains the single word translations and the rightmost cell at the top (the cell $(0, 5)$ in Figure 2) contains the translations for the whole sentence. Spans for which no items are available are marked with a dash “-”.

As a simple example, suppose that the decoder translated

Bob sends Alice a secret message
from English into the German sentence
Bob sendet Alice eine Geheimbotschaft

	0	1	2	3	4	5
0	Bob	-	-	-	-	-
1		sendet	-	-	-	-
2			Alice	-	-	-
3				eine	-	-
4					geheime	Geheimbotschaft
5						Nachricht

Table 2. A trivial example chart illustrating DIMwid’s chart display.

Table 2 shows a few (made-up) entries in the chart display for that translation. We omitted items for clarity. In the actual display the full translation should occur in cell (0,5). However, Table 2 shows that “*secret message*”, the span [5,6] of the source sentence, can be translated to “*Geheimbotschaft*”. Therefore the cell (4,5) of the chart contains this translation. In addition, the decoder could also use the translation into “*geheime Nachricht*” using the translations of cells (4,4) and (5,5).

The actual content of the items in a cell differs based on the trace format. For example, the items contain the source side in the level 3 verbosity output for phrase-based stack decoding. The Moses syntax-based chart decoder typically only outputs the terminals and the nonterminals of the target-side of the rule. As a minimum, the span of the source, which is translated, and its translation are always shown. The additional information depends on the trace format and is very specific and confusing to non-expert users of Moses. We decided to preserve this information for expert users, but most users can probably ignore the additional information. As illustrated in Section 4, it is very simple to adjust this behavior to accommodate any special needs.

4. Development

As mentioned earlier, we selected PYTHON and QT with straightforward adjustability in mind. PYTHON code is usually easy to read, runs on all major operating systems, and is very common in the programming community. The graphical framework QT is also freely available for all major operating systems, very common, and has PYTHON bindings, which allows us to exclusively use PYTHON for DIMwid.

4.1. Structure

DIMwid consists of three PYTHON source files. DIMwid.py creates the application using the interface designed in DIMterface.py, which sets up the classes related to the graphical user interface. Finally, DIMputs.py contains all the classes that represent the different input formats and provides functions for reading those formats.

4.2. Hacking

The simplicity of *DIMwid* allows for quick changes in the code. It can easily be adjusted to display any kind of output sorted into spans. No inside knowledge about PYTHON or the QT framework is required. Let us illustrate this by showing the steps needed to support a new input format. First, we add a class to `DIMputs.py` for the text-format that we want to load. We can follow the example of the other classes in `DIMputs.py`. In order to use our new class, the `DataInput`-class needs a function which reads the new format and stores the contained information into an object of the newly created class. At this point the basic functionality is present, but we still need to enable the new format in the graphical user interface. This is achieved by the following steps:

1. add a new format button to the *Format-Buttons-Drop-Down-Menu*,
2. create a new *WidgetAction* corresponding to this button,
3. connect the *WidgetAction* with the drop-down menu,
4. create a function that sets the `MainWindow`'s format to the new format,
5. connect the button to the format-setting function, and
6. add the new format to the `setPath`-function's *if-else*-block.

Since these code blocks exist for the natively supported formats, even non-experts should be able to perform these changes with the help of simple copy & paste actions.

5. Conclusion and Future Work

Our primary goal during the development of *DIMwid* was to make the analysis of the translation process easier. Such an analysis is beneficial for translation engineers that want to improve their system and to instructors that want to demonstrate the workings of the decoders in class. The graphical user interface of *DIMwid* displays the information of the typically huge and hardly readable trace files in an accessible manner. Currently, *DIMwid* supports (all) the trace outputs of both the phrase-based stack decoder and the syntax-based chart decoder of the Moses framework. The trace is uniformly presented in a chart, so all reported information is associated to a chart cell based on the covered span. Although *DIMwid* was developed for the traces of the Moses framework, it can easily be extended to read outputs of other frameworks (such as JOSHUA).

At present, *DIMwid* shows all items in the order that they occur in the traces. In future versions, we plan to combine the utility of the standard and full trace by highlighting the items that contribute to the best translation in the display of the full trace. In addition, we plan to add a format auto-detection that would remove the need to manually select a format. Ideally, we would also be able to graphically link items to their constituting items (i.e., the subspan items that were combined to form the current item). However, this feature has to be carefully implemented as it requires addi-

tional processing of the trace file and can thus potentially lead to major slow-down of *DIMwid*.

Acknowledgements

All authors were financially supported by the German Research Foundation (DFG) grant MA / 4959 / 1-1.

Bibliography

- Braune, Fabienne, Andreas Maletti, Daniel Quernheim, and Nina Seemann. Shallow local multi bottom-up tree transducers in statistical machine translation. In *Proc. ACL*, pages 811–821. Association for Computational Linguistics, 2013.
- Chiang, David. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.
- Hoang, Hieu, Philipp Koehn, and Adam Lopez. A uniform framework for phrase-based, hierarchical and syntax-based machine translation. In *Proc. IWSLT*, pages 152–159, 2009.
- Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proc. NAACL*, pages 48–54. Association for Computational Linguistics, 2003.
- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL*, pages 177–180. Association for Computational Linguistics, 2007. Demonstration session.
- Li, Zhifei, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren N. G. Thornton, Jonathan Weese, and Omar F. Zaidan. Joshua: an open source toolkit for parsing-based machine translation. In *Proc. WMT*, pages 135–139. Association for Computational Linguistics, 2009.
- Weese, Jonathan and Chris Callison-Burch. Visualizing data structures in parsing-based machine translation. *The Prague Bulletin of Mathematical Linguistics*, 93:127–136, 2009.

Address for correspondence:

Robin Kurtz

kurtzrn@ims.uni-stuttgart.de

Universität Stuttgart

Institut für Maschinelle Sprachverarbeitung

Pfaffenwaldring 5b, D-70569 Stuttgart, Germany



The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013 51-62

morphogen: Translation into Morphologically Rich Languages with Synthetic Phrases

Eva Schlinger, Victor Chahuneau, Chris Dyer

Language Technologies Institute, Carnegie Mellon University

Abstract

We present *morphogen*, a tool for improving translation into morphologically rich languages with synthetic phrases. We approach the problem of translating into morphologically rich languages in two phases. First, an inflection model is learned to predict target word inflections from source side context. Then this model is used to create additional sentence specific translation phrases. These “synthetic phrases” augment the standard translation grammars and decoding proceeds normally with a standard translation model. We present an open source Python implementation of our method, as well as a method of obtaining an unsupervised morphological analysis of the target language when no supervised analyzer is available.

1. Introduction

Machine translation into morphologically rich languages is challenging, due to lexical sparsity on account of grammatical features being expressed with morphology. In this paper, we present an open-source Python tool, *morphogen*, that leverages target language morphological grammars (either hand-crafted or learned unsupervisedly) to enable prediction of highly inflected word forms from rich, source language syntactic information.¹

Unlike previous approaches to translation into morphologically rich languages, our tool constructs sentence-specific translation grammars (i.e., phrase tables) for each sentence that is to be translated, but then uses a standard decoder to generate the final

¹<https://github.com/eschling/morphogen>

translation with no post-processing. The advantages of our approach are: (i) newly synthesized forms are highly targeted to a specific translation context; (ii) multiple alternatives can be generated with the final choice among rules left to a standard sentence-level translation model; (iii) our technique requires virtually no language-specific engineering; and (iv) we can generate forms that were not observed in the bilingual training data.

This paper is structured as follows. We first describe our “translate-and-inflect” model that is used to synthesize the target side of lexical translations rule given its source and its source context (§2). This model discriminates between inflectional options for predicted stems, and the set of inflectional possibilities is determined by a morphological grammar. To obtain this morphological grammar, the user may either provide a morphologically analyzed version of their target language training data, or a simple unsupervised morphology learner can be used instead (§3). With the morphologically analyzed parallel data, the parameters of the discriminative model are trained from the complete parallel training data using an efficient optimization procedure that does not require a decoder.

At test time, our tool creates **synthetic phrases** representing likely inflections of likely stem translations for each sentence (§4). We briefly present the results of our system on English–Russian, –Hebrew, and –Swahili translation tasks (§5), and then describe our open source implementation, and discuss how to use it with both user-provided morphological analyses and those of our unsupervised morphological analyzer² (§6).

2. Translate-and-Inflect Model

The task of the translate-and-inflect model is illustrated in Figure 1 for an English–Russian sentence pair. The input is a sentence e in the source language³ together with any available linguistic analysis of e (e.g., its dependency parse). The output f consists of (i) a sequence of stems, each denoted σ , and (ii) one morphological inflection pattern for each stem, denoted μ .⁴ Throughout, we use Ω_σ to denote the set of possible morphological inflection patterns for a given stem σ . Ω_σ might be defined by

²Further documentation is available in the morphogen repository.

³In this paper, the source language is always English. We use e to denote the source language (rather than the target language), to emphasize the fact that we are translating *from* a morphologically impoverished language *to* a morphologically rich one.

⁴When the information is available from the morphological analyzer, a stem σ is represented as a tuple of a lemma and its inflectional class.

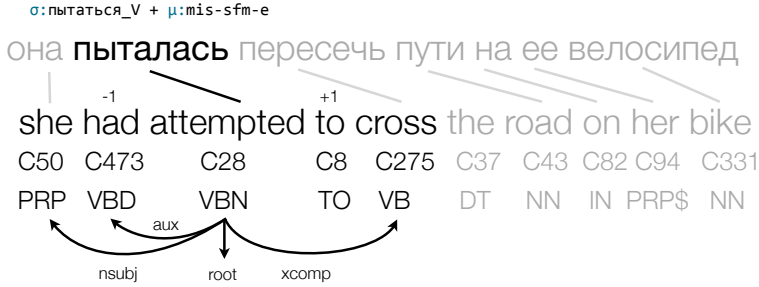


Figure 1. The inflection model predicts a form for the target verb stem based on its source attempted and the linear and syntactic source context. The inflection pattern mis-sfm-e (main+indicative+past+singular+feminine+medial+perfective) is that of a supervised analyzer.

a grammar; our models restrict Ω_σ to be the set of inflections observed anywhere in our monolingual or bilingual training data as a realization of σ .⁵

We define a probabilistic model over target words f . The model assumes independence between each target word f conditioned on the source sentence e and its aligned position i in this sentence.⁶ This assumption is further relaxed in §4 when the model is integrated in the translation system. The probability of generating each target word f is decomposed as follows:

$$p(f | e, i) = \sum_{\sigma * \mu = f} \underbrace{p(\sigma | e_i)}_{\text{gen. stem}} \times \underbrace{p(\mu | \sigma, e, i)}_{\text{gen. inflection}}.$$

Here, each stem is generated independently from a single aligned source word e_i , but in practice we use a standard phrase-based model to generate sequences of stems and only the inflection model operates word-by-word.

2.1. Modeling Inflection

In morphologically rich languages, each stem may be combined with one or more inflectional morphemes to express different grammatical features (e.g., case, definiteness, etc.). Since the inflectional morphology of a word generally expresses multiple features, we use a model that uses overlapping features in its representation of both

⁵This is a practical decision that prevents the model from generating words that would be difficult for a closed-vocabulary language model to reliably score. When open-vocabulary language models are available, this restriction can easily be relaxed.

⁶This is the same assumption that Brown et al. (1993) make in, for example, IBM Model 1.

$\left\{ \begin{array}{l} \text{source aligned word } e_i \\ \text{parent word } e_{\pi_i} \text{ with its dependency } \pi_i \rightarrow i \\ \text{all children } e_j \mid \pi_j = i \text{ with their dependency } i \rightarrow j \\ \text{source words } e_{i-1} \text{ and } e_{i+1} \\ \text{-- are } e_i, e_{\pi_i} \text{ at the root of the dependency tree?} \\ \text{-- number of children, siblings of } e_i \end{array} \right\}$	$\left\{ \begin{array}{l} \text{token} \\ \text{part-of-speech tag} \\ \text{word cluster} \end{array} \right\}$
---	--

Table 1. Source features $\varphi(e, i)$ extracted from e and its linguistic analysis. π_i denotes the parent of the token in position i in the dependency tree and $\pi_i \rightarrow i$ the typed dependency link.

the input (i.e., conditioning context) and output (i.e., the inflection pattern):

$$p(\mu \mid \sigma, e, i) = \frac{\exp [\varphi(e, i)^\top \mathbf{W} \psi(\mu) + \psi(\mu)^\top \mathbf{V} \psi(\mu)]}{\sum_{\mu' \in \Omega_\sigma} \exp [\varphi(e, i)^\top \mathbf{W} \psi(\mu') + \psi(\mu')^\top \mathbf{V} \psi(\mu')]} \quad (1)$$

Here, φ is an m -dimensional *source context* feature vector function, ψ is an n -dimensional *morphology* feature vector function, \mathbf{W} is an $m \times n$ parameter matrix, and \mathbf{V} is an $n \times n$ parameter matrix. In our implementation, φ and ψ return sparse vectors of binary indicator features, but other features can easily be incorporated.

2.2. Source Contextual Features: $\varphi(e, i)$

In order to select the best inflection of a target-language word, given the source word it translates from and the *context* of that source word, we seek to leverage numerous features of the context to capture the diversity of possible grammatical relations that might be encoded in the target language morphology. Consider the example shown in Figure 1, where most of the inflection features of the Russian word (past tense, singular number, and feminine gender) can be inferred from the context of the source word it is aligned to. To access this information, our tool uses parsers and other linguistic analyzers.

By default, we assume that English is the source language and provide wrappers for external tools to generate the following linguistic analyses of each input sentence:

- Part-of-speech tagging with a CRF tagger trained on sections 02–21 of the Penn Treebank,
- Dependency parsing with TurboParser (Martins et al., 2010), and
- Mapping of the tokens to one of 600 Brown clusters trained from 8B words of English text.⁷

⁷The entire monolingual data available for the translation task of the 8th ACL Workshop on Statistical Machine Translation was used. These clusters are available at <http://www.ark.cs.cmu.edu/cdyer/en-c600.gz>

From these analyses we then extract features from e by considering the aligned source word e_i , its preceding and following words, and its dependency neighbors. These are detailed in Table 1 and can be easily modified to include different features or for different source languages.

3. Morphological Grammars and Features

The discriminative model in the previous section selects an inflectional pattern for each candidate stem. In this section, we discuss where the inventory of possible inflectional patterns it will consider come from.

3.1. Supervised Morphology

If a target language morphological analyzer is available that analyses each word in the target of the bitext and monolingual training data into a stem and vector of grammatical features, the inflectional vector may be used directly to define $\psi(\mu)$ by defining a binary feature for each key-value pair (e.g., Tense=past) composing the tag. Prior to running morphogen, the full monolingual and target side bilingual training data should be analyzed.

3.2. Unsupervised Morphology

Supervised morphological analyzers that map between inflected word forms and abstract grammatical feature representations (e.g., PAST+SING) are not available for every language into which we might seek to translate. We therefore provide an *unsupervised* model of morphology that segments words into sequences of morphemes, assuming a concatenative generation process and a single analysis per type. To do so, we assume that each word can be decomposed into any number of prefixes, a stem, and any number of suffixes. Formally, we let M represent the set of all possible morphemes and define a regular grammar M^*MM^* (i.e., zero or more prefixes, a stem, and zero or more suffixes). We learn weights for this grammar by assuming that the probability of each prefix, stem, and suffix is given by a draw from a Dirichlet distribution over all morphemes and then inferring the most likely analysis.

Hyperparameters. To run the unsupervised analyzer, it is necessary to specify the Dirichlet hyperparameters ($\alpha_p, \alpha_s, \alpha_t$) which control the sparsity of the inferred prefix, stem, and suffix lexicons, respectively. The learned morphological grammar is (rather unfortunately) very sensitive to these settings, and some exploration is necessary. As a rule of thumb, we observe that $\alpha_p, \alpha_s \ll \alpha_t \ll 1$ is necessary to recover useful segmentations, as this encodes that there are many more possible stems than inflectional affixes; however the absolute magnitude will depend on a variety of factors. Default values are $\alpha_p = \alpha_s = 10^{-6}$, $\alpha_t = 10^{-4}$; these may be adjusted by factors of 10 (larger to increase sparsity; smaller to decrease it).

Unsupervised morphology features: $\psi(\mu)$ For the unsupervised analyzer, we do not have a mapping from morphemes to grammatical features (e.g., +past); however, we can create features from the affix sequences obtained after morphological segmentation. We produce binary features corresponding to the content of each potential affixation position relative to the stem. For example, the unsupervised analysis $wa+ki+wa+STEM$ of the Swahili word *wakiwapiga* will produce the following features:

Prefix[-3][wa] Prefix[-2][ki] Prefix[-1][wa].

3.3. Inflection Model Parameter Estimation

From the analyzed parallel corpus (source side syntax and target side morphological analysis), morphogen sets the parameters \mathbf{W} and \mathbf{V} of the inflection prediction model (Eq. 1) using stochastic gradient descent to maximize the conditional log-likelihood of a training set consisting of pairs of source sentence contextual features (φ) and target word inflectional features (ψ). The training instances are word alignment pairs from the full training corpus. When morphological category information is available, an independent model may be trained for each open-class category (e.g., nouns, verbs); but, by default a single model is used for all words (excluding words shorter than a minimum length).

It is important to note here that our richly parameterized model is trained on the *full parallel training corpus*, not just on the small number of development sentences. This is feasible because, in contrast to standard discriminative translation models which seek to discriminate good complete translations from bad complete translations, morphogen’s model must only predict how good each possible *inflection* of an independently generated stem is. All experiments reported in this paper used models trained on a single processor using a Cython implementation of the SGD optimizer.⁸

4. Synthetic Phrases

How is morphogen used to improve translation? Rather than using the translate-and-inflect model directly to perform translation, we use it just to augment the set of rules available to a conventional hierarchical phrase-based translation model (Chiang, 2007; Dyer et al., 2010). We refer to the phrases it produces as **synthetic phrases**. The aggregate grammar consists of both synthetic and “default” phrases and is used by an unmodified decoder.

The process works as follows. We use the suffix-array grammar extractor of Lopez (2007) to generate sentence-specific grammars from the fully inflected version of the training data (the default grammar) and also from the stemmed variant of the training

⁸For our largest model, trained on 3.3M Russian words, $n = 231K * m = 336$ feature were produced, and 10 SGD iterations at a rate of 0.01 were performed in less than 16 hours.

Russian supervised	Hebrew	Swahili
Verb: 1st Person child(nsubj)=I child(nsubj)=we	Suffix ׁ (masculine plural) parent=NNS after=NNS	Prefix <i>li</i> (past) source=VBD source=VBN
Verb: Future tense child(aux)=MD child(aux)=will	Prefix ׀ (first person sing. + future) child(nsubj)=I child(aux)='ll	Prefix <i>nita</i> (1st person sing. + future) child(aux) child(nsubj)=I
Noun: Animate source=animals/victims/...	Prefix ׃ (preposition like/as) child(pre)=IN parent=as	Prefix <i>ana</i> (3rd person sing. + present) source=VBZ
Noun: Feminine gender source=obama/economy/...	Suffix ׳ (possessive mark) before=my child(poss)=my	Prefix <i>wa</i> (3rd person plural) before=they child(nsubj)=NNS
Noun: Dative case parent(iobj)	Suffix ן (feminine mark) child(nsubj)=she before=she	Suffix <i>tu</i> (1st person plural) child(nsubj)=she before=she
Adjective: Genitive case grandparent(poss)	Prefix ׁ׃ (when) before=when before=WRB	Prefix <i>ha</i> (negative tense) source=no after=not

Figure 2. Examples of highly weighted features learned by the inflection model. We selected a few frequent morphological features and show their top corresponding source context features.

data (the stemmed grammar). We then extract a set of translation rules that only contain terminal symbols (sometimes called “lexical rules”) from the stemmed grammar. The (stemmed) target side of each such phrase is then *re-inflected* using the inflection model described above (§2), conditioned on the source sentence and its context. Each stem is given its most likely inflection. The resulting rules are added to the default grammar for the sentence to produce the aggregate grammar.

The standard translation rule features present on the stemmed grammar rules are preserved, and morphogen adds the following features to help the decoder select good synthetic phrases: (i) a binary feature indicating that the phrase is synthetic; (ii) the log probability of the inflected form according to the inflection model; and (iii) if available, counts of the morphological categories inflected.

5. Experiments

We briefly report in this section on some experimental results obtained with our tool. We ran experiments on a 150k sentence Russian–English task (WMT2013; news-commentary), a 134k sentence English–Hebrew task (WIT³ TED talks corpus), and a 15k sentence English–Swahili Task. Space precludes a full discussion of the performance of the classifier,⁹ but we can also inspect the weights learned by the model to assess the effectiveness of the features in relating source-context structure with target-side morphology. Such an analysis is presented in Figure 2.

⁹We present our approach and the results of both the intrinsic and extrinsic evaluations in much more depth in Chahuneau et al. (in review)

	EN→RU	EN→HE	EN→SW
Baseline	14.7±0.1	15.8±0.3	18.3±0.1
+Class LM	15.7±0.1	16.8±0.4	18.7±0.2
+Synthetic			
unsupervised	16.2±0.1	17.6±0.1	19.0±0.1
supervised	16.7±0.1	—	—

Table 2. Translation quality (measured by bleu) averaged over 3 MIRA runs.

5.1. Translation

We evaluate our approach in the standard discriminative MT framework. We use cdec (Dyer et al., 2010) as our decoder and perform MIRA training (Chiang, 2012) to learn feature weights. We compare the following configurations:

- A baseline system, using a 4-gram language model trained on the entire monolingual and bilingual data available.
- An enriched system with a class-based n-gram language model¹⁰ trained on the monolingual data mapped to 600 Brown clusters. Class-based language modeling is a strong baseline for scenarios with high out-of-vocabulary rates but in which large amounts of monolingual target-language data are available.
- The enriched system further augmented with our inflected synthetic phrases. We expect the class-based language model to be especially helpful here and capture some basic agreement patterns that can be learned more easily on dense clusters than from plain word sequences.

We evaluate translation quality by translating and measuring BLEU on a held-out evaluation corpus, averaging the results over 3 MIRA runs (Table 2). For all languages, using class language models improves over the baseline. When synthetic phrases are added, significant additional improvements are obtained. For the English–Russian language pair, where both supervised and unsupervised analyses can be obtained, we notice that expert-crafted morphological analyzers are more efficient at improving translation quality.

6. Morphogen Implementation Discussion and User’s Guide

This section describes the open-source Python implementation of this work, morphogen.¹¹ Our decision to use Python means the code—from feature extraction to grammar processing—is generally readable and simple to modify for research purposes. For example, with few changes to the code, it is easy to expand the number of

¹⁰For Swahili and Hebrew, $n = 6$; for Russian, $n = 7$.

¹¹<https://github.com/eschling/morphogen>

synthetic phrases created by generating k-best inflections (rather than just the most probable inflection), or to restrict the phrases created based on some source side criterion such as type frequency, POS type, or the like.

Since there are many processing steps that must be coordinated to run `morphogen`, we provide reference workflows using `ducttape`¹² for both supervised and unsupervised morphological analyses (discussed below). While these workflows are set up to be used with `cdec`, `morphogen` generates grammars that could be used with any decoder that supports per-sentence grammars. The source language processing, which we do for English using `TurboParser` and `TurboTagger`, could be done with any tagger and any parser that can produce basic Stanford dependencies. The source language does not necessarily need to be English, although our approach depends on having detailed source side contextual information.¹³

We now review the steps that must be taken to run `morphogen` with either an external (generally supervised) morphological analyzer or the unsupervised morphological analyzer we described above. These steps are implemented in the provided `ducttape` workflows.

Running `morphogen` with an external morphological analyzer. If a supervised morphological analyzer is used, the parallel training data must be analyzed on the target side, with each line containing four fields (source sentence, target sentence, target stem sentence, target analysis sequence), where fields are separated with the triple pipe (`|||`) symbol. Target language monolingual data must likewise be analyzed and provided in a file where each line contains three fields (sentence, stem sentence, analysis sequence) and separated by triple pipes. For supervised morphological analyses, the user must also provide a python configuration file that contains a function `get_attributes`,¹⁴ which parses the string representing the target morphological analysis into a set of features that will be exposed to the model as the target morphological feature vector $\psi(\mu)$.

Running `morphogen` with the unsupervised morphological analyzer. To use unsupervised morphological analysis, two additional steps (in addition to those required for an external analyzer) are required:

¹²`ducttape` is an open-source workflow management system similar to `make`, but designed for research environments. It is available from <https://github.com/jhclark/ducttape>.

¹³It is also unclear how effective our model would be when translating between two morphologically rich languages, since we assume that the source language expresses syntactically many of the things which the target language expresses with morphology. This is a topic for future research, and one that will be facilitated by `morphogen`.

¹⁴See the `morphogen` documentation for more information on defining this function. The configuration for the Russian positional tagset used for the “supervised” Russian experiments is provided as an example.

```

Tokenized source: We 've heard that empty promise before . ||| ↩
Tokenized target (inflected): Но мы и раньше слышали эти пустые обещания . ||| ↩
Tokenized target (stemmed): но мы и раньше слышать этот пустой обещание . ||| ↩
POS + inflectional features: C P-1-pnn C R Vmis-p-a-e P---paa Afmpaf Ncnpan .

```

Figure 3. Example supervised input; arrows indicate that the text wraps around to the next line just for ease of reading (there should be no newline character in the input).

- use `fast_umorph`¹⁵ to get unsupervised morphological analyses (see §3.2);
- use `seg_tags.py` with these segmentations to retrieve the lemmatized and tagged version of the target text. Tags for unsupervised morphological segmentations are a simple representation of the learned segmentation. Words less than four characters are tagged with an X and subsequently ignored.

Remaining training steps. Once the training data has been morphologically analyzed, the following steps are necessary:

- process the source side of the parallel data using TurboTagger, TurboParser, and Brown clusters.
- use `lex_align.py` to extract parallel source and target stems with category information. This lemmatized target side is used with cdec’s `fast_align` to produce alignments.
- combine to get fully preprocessed parallel data, in the form (source sentence, source POS sequence, source dependency tree, source class sequence, target sentence, target stem sequence, target morphological tag sequence, word alignment), separated by the triple pipe.
- use `rev_map.py` to create a mapping from (stem, category) to sets of possible inflected forms and their tags. Optionally, monolingual data can be added to this mapping, to allow for the creation of inflected word forms that appear in the monolingual data but not in the parallel training data. If a (stem, category) pair maps to multiple inflections that have the same morphological analysis, the most frequent form is used.¹⁶
- train structured inflection models with SGD using `struct_train.py`. A separate inflection model must be created for each word category that is to be inflected. There is only a single category when unsupervised segmentation is used.

¹⁵https://github.com/vchahun/fast_umorph

¹⁶This is only possible when a supervised morphological analyzer is used, as our unsupervised tags are just a representation of the segmentation (e.g. `wa+ku+STEM`).

Using morphogen for tuning and testing. At tuning and testing time, the following steps are run:

- extract two sets of per-sentence grammars, one with the original target side and the other with the lemmatized target side
- use the extracted grammars, the trained inflection models, and the reverse inflection map with `synthetic_grammar.py` to create an augmented grammar that consists of both the original grammar rules and any inflected synthetic rules (§4). By default, only the single best inflection is used to create a synthetic rule, but this can be modified easily.
- add target language model and optionally a target class based language model. Proceed with decoding as normal (we tune with MIRA and then evaluate on our test set)

Using the ducttape workflows. The provided ducttape workflows implement the above pipelines, including downloading all of the necessary tool dependencies so as to make the process as simple as possible. The user simply needs to replace the global variables for the dev, test, and training sets with the correct information, point it at their version of morphogen, and decide which options they would like to use. Sample workflow paths are already created (e.g. path with/without Monolingual training data, with/without class based target language model). These can be modified as needed.

Analysis tools. We also provide the scripts `predict.py` and `show_model.py`. The former is used to perform an intrinsic evaluation of the inflection model on held out development data. The latter provides a detailed view of the top features for various inflections, allowing for manual inspection of the model as in Figure 2. An example workflow script for the intrinsic evaluation is also provided.

7. Conclusion

We have presented an efficient technique which exploits morphologically analyzed corpora to produce new inflections possibly unseen in the bilingual training data and described a simple, open source tool that implements it. Our method decomposes into two simple independent steps involving well-understood discriminative models.

By relying on source-side context to generate additional local translation options and by leaving the choice of the global sentence translation to the decoder, we sidestep the issue of inflecting imperfect translations and we are able to exploit rich annotations to select appropriate inflections without modifying the decoding process or even requiring that a specific decoder or translation model type be used.

We also achieve language independence by exploiting unsupervised morphological segmentations in the absence of linguistically informed morphological analyses, making this tool appropriate for low-resource scenarios.

Acknowledgments

This work was supported by the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-10-1-0533. We would like to thank Kim Spasaro for curating the Swahili development and test sets.

Bibliography

- Brown, Peter F., Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.
- Chahuneau, Victor, Eva Schlinger, Chris Dyer, and Noah A. Smith. Translating into morphologically rich languages with synthetic phrases, in review.
- Chiang, David. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.
- Chiang, David. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13:1159–1187, 2012.
- Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proc. of ACL*, 2010.
- Lopez, Adam. Hierarchical phrase-based translation with suffix arrays. In *Proc. of EMNLP*, 2007.
- Martins, André F.T., Noah A. Smith, Eric P. Xing, Pedro M.Q. Aguiar, and Mário A.T. Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *Proc. of EMNLP*, 2010.

Address for correspondence:

Eva Schlinger
eva@cmu.edu
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA



The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013 63-72

RankEval: Open Tool for Evaluation of Machine-Learned Ranking

Eleftherios Avramidis

Language Technology Lab
German Research Center for Artificial Intelligence (DFKI)

Abstract

Recent research and applications for evaluation and quality estimation of Machine Translation require statistical measures for comparing machine-predicted ranking against gold sets annotated by humans. Additional to the existing practice of measuring segment-level correlation with Kendall tau, we propose using ranking metrics from the research field of Information Retrieval such as Mean Reciprocal Rank, Normalized Discounted Cumulative Gain and Expected Reciprocal Rank. These reward systems that predict correctly the highest ranked items than the one of lower ones. We present an open source tool "RANKEval" providing implementation of these metrics. It can be either run independently as a script supporting common formats or can be imported to any Python application.

1. Introduction

Research in Machine Translation (MT) has resulted into the development of various Machine Translation systems over the years. One of the most prominent ways of assessing their performance is to do it comparatively, i.e. comparing them and ordering them in terms of quality. This offers the possibility to be consistent with human quality judgements, without having to rely on underspecified "absolute" quality numbers, which are often hard to define and derive objectively.

The result of ordering translations in terms of their quality has had a few applications focusing on a sentence level. One of these applications refers to assessing the quality of automatic evaluation metrics. In particular, since quite a few years, the Evaluation Shared Task of the Workshop on Machine Translation (Callison-Burch et al., 2008) has used the so-called "segment-level" ranking, in order to compare rank-

ings produced by automatic evaluation metrics against the ones devised by human annotators. In most cases, translation segments are defined by periods, roughly as long as one sentence.

Additionally, the use of several Machine Translation (MT) systems within translation workflows pretty often requires automatic Quality Estimation systems that predict the ranking of the translation quality on a sentence level. The performance of such Quality Estimation rankers can be assessed when sentence-level ranking lists are compared with the ranking a human would do.

In both above tasks, predicted ranking is evaluated against the human ranking, using calculations following the Kendall tau correlation coefficient. On top of that, in this paper we present some existing measures that have been used in other fields, but are suitable for tasks relative to Machine Translation, such as the ones described above. The measures are wrapped in an open source tool called RANKEVAL which is described in detailed.

2. Previous Work

The simplest measure of its kind, tau, was introduced by Kendall (1938) with the purpose to analyze experiments on psychology, where the order given by different observers is compared. This measure has been analyzed and modified over the years for several purposes (Knight, 1966; Agresti, 1996; Christensen, 2005) and has been also applied to text technologies (Lapata, 2003; Cao et al., 2007). Since 2008 it appears modified as an official segment-level measure for the evaluation metrics in the yearly shared task for Machine Translation (Callison-Burch et al., 2008). This is the reason we decided to re-implement Kendall tau with penalization of ties, although there is already another open source version by SciPy (Oliphant, 2007), however with different accounting of ties.

More metrics emerged for use with Information Retrieval. Directed Cumulated Gain (Järvelin and Kekäläinen, 2002) was extended to the measures of Discounted Cumulative Gain, Ideal Cumulative Gain and Normalized Cumulative Gain (Wang et al., 2013). Mean Reciprocal Rank was introduced as an official evaluation metric of TREC-8 Shared Task on Question Answering (Radev et al., 2002) and has also been applied successfully for the purpose of evaluating MT n-best lists and transliteration in the frame of the yearly Named Entities Workshop (Li et al., 2009). Additionally, Expected Reciprocal Rank (Chapelle et al., 2009) was optimized for Search Engine results and used as a measure for a state-of-the-art *Learning to Rank* challenge (Chapelle and Chang, 2011).

In the following sections we present shortly the evaluation measures and the way they have been implemented to suit the evaluation needs of MT.

3. Methods

In a ranking task, each translation is assigned an integer (further called a *rank*), which indicates its quality as compared to the competing translations for the same source sentence. E.g. given one source sentence and n translations for it, each of the latter would get a rank in the range $[1, n]$. The aim of the methods below is to produce a score that indicates the quality of an automatically predicted ranking against human rankings.

3.1. Kendall's Tau

3.1.1. Original calculation

Kendall's tau (Kendall, 1938; Knight, 1966) measures the correlation between two ranking lists on a segment level by counting *concordant* or *discordant* pairwise comparisons: For every sentence, the two rankings (machine-predicted and human) are first decomposed into pairwise comparisons. Then, a concordant pair is counted when each predicted pairwise comparison matches the respective pairwise comparison by the human annotator; otherwise a discordant pair is counted. Consequently, tau is computed by:

$$\tau = \frac{\text{concordant} - \text{discordant}}{\text{concordant} + \text{discordant}} \quad (1)$$

with values ranging between minus one and one. The closer $|\tau|$ values get to one, the better the ranking is. In particular, when values get close to minus one, the ranking is also good, but the order of its element should be reversed. This is typical for evaluation metrics which assign higher scores to better translations, whereas humans evaluations usually assign lower ranks to the better ones. A value of zero indicates no correlation.

3.1.2. Penalization of ties

A common issue in ranking related to MT is that the same rank may be assigned to two or more translation candidates, if the translations are of similar quality (i.e. there is no distinguishable difference between them). Such a case defines a *tie* between the two translation candidates. A tie can exist in both the gold-standard ranking (as a decision by an annotator based on his judgment) and the predicted ranking (as an uncertain decision by the machine ranker).

As one can see in the fraction of equation 1, ties are not included in the original calculation of tau, which may yield impropotional results when a ranker produces a huge amount of ties and only a few correct comparisons (as only the latter would be included in the denominator). Previous work includes a few tau extensions to address this issue (Degenne, 1972). We focus on the ties penalization of Callison-Burch et al. (2008) which follows these steps:

- Pairwise ties in the human-annotated test set are excluded from the calculations, as ties are considered to form uncertain samples that cannot be used for evaluation.
- For each remaining pairwise comparison, where human annotation has not resulted in a tie, every tie on the machine-predicted rankings is penalized by being counted as a discordant pair.

$$\tau = \frac{\text{concordant} - (\text{discordant} + \text{ties})}{\text{concordant} + \text{discordant} + \text{ties}} \quad (2)$$

With these modifications, the values of the ratio are still between minus one and one, but since a ties penalty has been added, values close to minus one can no longer be considered as a good result and if needed, ranks must be reverted prior to the calculation.

3.1.3. Segment-level correlation on a document level

As the above calculation is defined on a segment (sentence) level, we accumulate tau on the data set level in two ways:

- **Micro-averaged tau** (τ_μ) where concordant and discordant counts from all segments (i.e., sentences) are gathered and the fraction is calculated with their sums.¹
- **Macro-averaged tau** (τ_m) where tau is calculated on a segment level and then averaged over the number of sentences. This shows equal importance to each sentence, irrelevant of the number of alternative translations.

3.1.4. P-value for Kendall tau

For an amount of n ranked items, we calculate the two-sided p-value for a hypothesis test whose null hypothesis is an absence of association (Oliphant, 2007):

$$z = \frac{\tau}{\sqrt{\frac{4n+10}{9n(n-1)}}} \quad (3)$$

$$p = \text{erfc} \left(\frac{|z|}{\sqrt{2}} \right) \quad (4)$$

where erfc is the complementary error function of the fraction.

3.2. First Answer Reciprocal Rank and Mean Reciprocal Rank

Kendall tau correlation sets the focus on the entire ranking list, giving an equal weight to the correct prediction of all ranks. Another set of measures emphasizes only

¹ τ_μ is the tau calculation that appears in WMT results

on the best item(s) (according to the humans) and how high they have been ranked by the ranker, assuming that our interest for the worse items is less. The first measure of this kind the First Answer Reciprocal Rank (FARR) which is the multiplicative inverse of the rank of the first correct answer (Radev et al., 2002), having an index i :

$$\text{FARR} = \frac{1}{\text{rank}_i} \quad (5)$$

A common use of FARR is through the Mean Reciprocal Rank, which averages the segment-level reciprocal ranks over all sentences:

$$\text{MRR} = \frac{1}{n} \sum_{j=1}^n \frac{1}{\text{rank}_{j,i}} \quad (6)$$

where n is the number of sentences, j the sentence index and $\text{rank}_{j,i}$ the rank of the first correct answer for this sentence. As FARR is calculated over only one rank, ties need only be considered only if they occur for this particular rank. In that case, we only consider the ranker's best prediction for it.

3.3. Cumulative Gain

This family of measures is based on Discounted Cumulative Gain (DCG), which is a weighted sum of the degree of relevance of the ranked items. This introduces a *discount*, which refers to the fact that the rank scores are weighted by a decreasing function of the rank i of the item.

$$\text{DCG}_p = \sum_{i=1}^p \frac{2^{\text{rel}_i} - 1}{\log_2(i + 1)} \quad (7)$$

In our case, we consider that relevance of each rank (rel_i) is inversely proportional to its rank index.

The most acknowledged measure of this family is the Normalized Discounted Cumulative Gain (NDCG), which divides the DCG by the Ideal Discounted Cumulative Gain (IDCG), the maximum possible DCG until position p . Then, NDGC is defined as:

$$\text{NDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p}. \quad (8)$$

3.4. Expected Reciprocal Rank

The Expected Reciprocal Rank (ERR) has been suggested as an improvement of NDCG in order to better model the fact that the likelihood a user examines the translation at rank i is dependent on how satisfied the user was with the translations observed previously in the ranking list (Chapelle et al., 2009), introducing the so-called *user cascade model*.

The probability of relevance is here given by

$$R_i = \frac{2^{\text{rel}_i} - 1}{2^{\text{rel}_{i_{\max}}} - 1} \quad (9)$$

and given that the user stops at position r , this forms the calculation of ERR as:

$$\text{ERR} = \sum_{r=1}^n \frac{1}{r} \prod_{i=1}^{r-1} (1 - R_i) R_r. \quad (10)$$

3.5. Simple Measures

Additionally to the above sophisticated measures, we also use simpler measures. These are:

- **Best predicted vs human (BPH):** For each sentence, the item selected as best by the machine ranker, may have been ranked lower by the humans. This measure returns a vector of how many times the item predicted as best has fallen into each of the human ranks.
- **Average predicted:** the average human rank of the item chosen by the machine ranker as best.

3.6. Normalization of Ranking Lists

Normalization emerges as a need from the fact that in practice there are many different ways to order items within the range of the rank values. This becomes obvious if one considers ties. Since there is no standard convention for ordering ties, the same list may be represented as [1, 2, 2, 3, 4], [1, 2, 2, 4, 5], [1, 3, 3, 4, 5] or even [1, 2.5, 2.5, 4, 5]. The alternative representations are even more when more ties are involved.

All representations above are equivalent, since there is no absolute meaning of quality in the values involved. Nevertheless, the rank value plays a role for the calculation of some of the metrics explained above. For this purpose, we consider several different normalization options of such ranking lists:

- **minimize:** reserves only one rank position for all tied items of the same rank (e.g.: [1, 2, 2, 3, 4]).
- **floor:** reserves all rank positions for all tied items of the same rank, but sets their value to the minimum tied rank position (e.g: [1, 2, 2, 4, 5]).
- **ceiling:** reserves all rank positions for all tied items of the same rank, but sets their value to the maximum tied rank position (e.g: [1, 3, 3, 4, 5]). This is the default setting, inline to many previous experiments.
- **middle:** reserves all rank positions for all tied items of the same rank, but sets their value to the middle of the tied rank positions (e.g: [1, 2.5, 2.5, 3, 4]).

4. Implementation

4.1. Coding and Architecture

The code has been written in Python 2.7, taking advantage of the easier calculation due to the dynamic assignment of items in the lists. Few functions from `numpy` and `scipy` libraries are included, which therefore sets them as prerequisites for running the tool. The code is available in an open `git` repository.²

The code includes one function for each ranking measure, with the exception of `NDGC` and `ERR` which are merged into one loop for saving computational time. Each function receives as parameters the predicted and the human (also referred to as *original*) rankings. Depending on how many the results of each function are, they are returned as single float values, tuples or `dict` structures, as explained in the documentation of each function. The code is organized in two Python modules, so that the functions can be imported and used by other Python programs.

- `ranking.segment`, where the segment-level calculation takes place, and
- `ranking.set`, where the segment-level calculations are aggregated to provide results for the entire data set. This mainly includes averaging (as explained previously) but also the simple measures (Section 3.5). There is also a utility function that executes all available functions and returns the results altogether.

The ranking lists are handled by the `sentence.ranking.Ranking` class, which includes the functions for normalizing the included values.

4.2. Stand-Alone Execution

A stand-alone execution is also possible using the command line script `rankeval.py` which resides on the root of the package. This script is responsible for reading command line parameters on the execution, opening and parsing the files with the ranking lists, starting the evaluation and displaying the results. The script supports reading two formats:

- a text-based format, similar to the one used for WMT Evaluation Shared Task
- an XML-based format, which includes the sentence-level ranking annotations along with the source and translated text. This format has been used in several quality estimation tasks

4.3. Linear Computation of ERR

Since the mathematical formula for the computation of the Expected Reciprocal Rank is computed in exponential time, we use the simplified computation suggested by Chapelle et al. (2009), which is outlined in Algorithm 1. The algorithm reduces the

²<https://github.com/lefterav/rankeval>

Algorithm 1: Linear computation of Expected Reciprocal Rank

foreach i *in* $[0, n]$ **do** $g_i \leftarrow \text{RelevanceGrade}(i)$ $p \leftarrow 1, \text{ERR} \leftarrow 0.$ **for** $r \leftarrow 1$ **to** n **do** $R \leftarrow \text{RelevanceProb}(g_r)$ $\text{ERR} \leftarrow \text{ERR} + p \times R/r$ $p \leftarrow p \times (1 - R)$ **return** ERR

computational perplexity by calculating the relevance grades g_i only once for each rank i . This is used during the loop for calculating the relevance probability R_i and gradually augmenting the ERR value.

5. Discussion

It is hard to evaluate new metrics, as we examine a meta-evaluation level, where there is no gold standard to compare with. Therefore, we leave this kind of evaluation to further work, as we hope that the tool will make it possible to apply the measures on different types of data.

As an indication of the correlation between the measures in a range of experiments we present a graphical representation (Figure 1) of all measure values, given for 78 quality estimation experiments on ranking. These experiments were done with various machine learning parametrizations (Avramidis, 2012) over the basic set-up of the Sentence Ranking Shared Task on Quality estimation (WMT13 – Bojar et al., 2013). The experiments are ordered based on their descending MRR score, which appears as a straight line, whereas the scores given by the other measures for the respective experiments are plotted with the rest of the lines.

Each measure has a different range of values, which means that the position on the Y axis, or the inclination are of no interest for the comparison. The interesting point are the fluctuations of each measure scores as compared to the others. As expected, we see that the measures of the same family seem to correlate with each other.

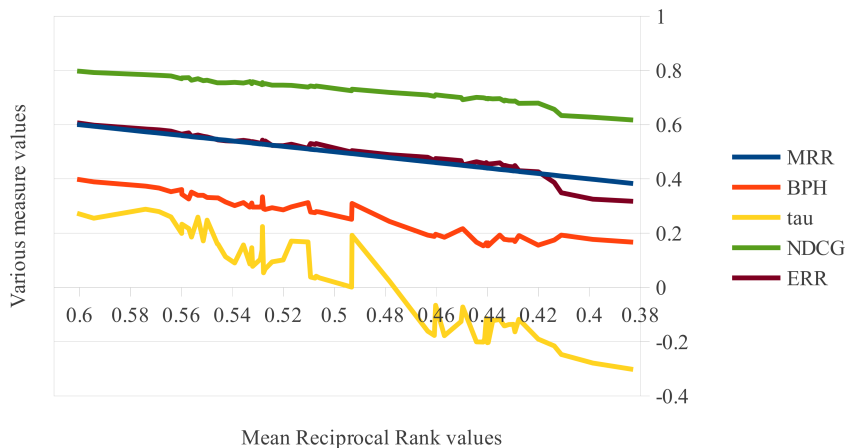


Figure 1. Plotting the values of the various measures (Y axis) for 78 quality estimation experiments ordered by descending MRR (X axis)

Acknowledgments

This work has been developed within the TaraXÜ project, financed by TSB Technologiestiftung Berlin – Zukunftsfonds Berlin, co-financed by the European Union – European fund for regional development. Many thanks to Prof. Hans Uszkoreit for the supervision, Dr. Aljoscha Burchardt, Dr. Maja Popović and Dr. David Vilar for their useful feedback.

Bibliography

- Agresti, Alan. *An introduction to categorical data analysis*, volume 135. Wiley New York, 1996.
- Avramidis, Eleftherios. Comparative quality estimation: Automatic sentence-level ranking of multiple machine translation outputs. In *Proceedings of 24th International Conference on Computational Linguistics*, pages 115–132, Mumbai, India, Dec. 2012. The COLING 2012 Organizing Committee.
- Bojar, Ondřej, Christian Buck, Chris Callison-Burch, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. Findings of the 2013 workshop on statistical machine translation. In *8th Workshop on Statistical Machine Translation*, Sofia, Bulgaria, 2013. Association for Computational Linguistics.
- Callison-Burch, Chris, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. Further meta-evaluation of machine translation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 70–106, Columbus, Ohio, June 2008. Association for Computational Linguistics.

- Cao, Zhe, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.
- Chapelle, Olivier and Yi Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research-Proceedings Track*, 14:1–24, 2011.
- Chapelle, Olivier, Donald Metzler, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on Information and knowledge management - CIKM '09*, page 621, New York, New York, USA, Nov. 2009. ACM Press. ISBN 9781605585123. doi: 10.1145/1645953.1646033.
- Christensen, David. Fast algorithms for the calculation of Kendall's τ . *Computational Statistics*, 20(1):51–62, 2005.
- Degenne, Alain. *Techniques ordinales en analyse des donn[ées]es statistique*. Classiques Hachette, 1972.
- Järvelin, Kalervo and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, Oct. 2002. ISSN 10468188. doi: 10.1145/582415.582418.
- Kendall, Maurice G. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938. doi: 10.1093/biomet/30.1-2.81.
- Knight, William R. A computer method for calculating kendalls tau with ungrouped data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.
- Lapata, Mirella. Probabilistic text structuring: Experiments with sentence ordering. In *Annual Meeting of the Association for Computational Linguistics*, pages 545–552, 2003.
- Li, Haizhou, A Kumaran, Vladimir Pervouchine, and Min Zhang. Report of NEWS 2009 machine transliteration shared task. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pages 1–18, Suntec, Singapore, Aug. 2009. Association for Computational Linguistics.
- Oliphant, Travis E. SciPy: Open source scientific tools for Python. *Computing in Science and Engineering*, 9(3):10–20, 2007. URL <http://www.scipy.org>.
- Radev, Dragomir, Hong Qi, Harris Wu, and Weiguo Fan. Evaluating web-based question answering systems. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, volume 1001, Las Palmas, Spain, 2002. European Language Resources Association (ELRA).
- Wang, Yining, Wang Liwei, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. A theoretical analysis of NDCG ranking measures. In *26th Annual Conference on Learning Theory*, 2013.

Address for correspondence:

Eleftherios Avramidis
eleftherios.avramidis@dfki.de
Language Technology Lab
German Research Center for Artificial Intelligence (DFKI)
Alt Moabit 91c, Berlin, Germany



XenC: An Open-Source Tool for Data Selection in Natural Language Processing

Anthony Rousseau

Laboratoire d'Informatique de l'Université du Maine (LIUM)

Abstract

In this paper we describe XenC, an open-source tool for data selection aimed at Natural Language Processing (NLP) in general and Statistical Machine Translation (SMT) or Automatic Speech Recognition (ASR) in particular. Usually, when building a SMT or ASR system, the considered task is related to a specific domain of application, like news articles or scientific talks for instance. The goal of XenC is to allow selection of relevant data regarding the considered task, which will be used to build the statistical models for such a system. It is done by computing the difference between cross-entropy scores of sentences from a large out-of-domain corpus and sentences from a corpus considered as in-domain for the task. Written in C++, this tool can operate on monolingual or bilingual data and is language-independent. XenC, now part of the LIUM toolchain for SMT, is actively developed since December 2011 and used in many MT projects.

1. Introduction

In Natural Language Processing, in general, and in Statistical Machine Translation or Automatic Speech Recognition, in particular, a system and its models are often considered as dynamic, always-evolving entities. These statistical models are not usually set in stone since they can be adapted to a target task or re-estimated with new or additional data. Also, their performance can be enhanced by various techniques, which can occur before, during or after the actual system processing. Among these, one of the most efficient pre-processing technique is data selection, i.e. the fact to carefully choose which data will be injected into the system we are going to build.

In this paper, while focusing on the Statistical Machine Translation field, we describe an open-source tool named XenC, which can be used to easily perform a data

selection for both monolingual data, aimed at Language Models (LM), and bilingual data, aimed at Translation Models (TM). This tool is freely available for both commercial and non-commercial use and is released under the GNU General Public License version 3.¹ Its most recent source code is accessible at: <https://github.com/rousseau-lium/XenC>.

The paper is organized as follows: in Section 2, we expose the motivations of our work on this tool. Section 3 describes the tool and the way it works. In Section 4, we present the requirements for the usage of the tool. Section 5 is dedicated to usage instructions, i.e. how to run XenC efficiently. In Section 6 we present some experimental results to illustrate the interest of such a tool. Then, Section 7 concludes this paper and expose some future plans for XenC.

2. Motivations

Most of the time, a translation system is built to fit a given task or a specific domain of application, like medical reports or court session transcriptions. This implies to dispose of a suitable corpus, which can be viewed as *in-domain*, reasonably large to produce an efficient system. Unfortunately, this is rarely the case, as most of the corpora sets usually available in SMT are quite generic and large quantities of relevant data for a desired task or domain are generally difficult to find. These corpora, not adapted for a particular task, can be viewed as *out-of-domain*. Moreover, another issue arising from using such generic corpora is that they can contain useless, or worse, harmful data for the models we want to estimate, thus lowering the translation quality.

With this in mind, the main idea behind XenC is to allow the extraction of relevant sentences (regarding the target translation task or domain) from an *out-of-domain corpus* by comparing them to the sentences of an *in-domain corpus*. Based on previous theoretical work by Moore and Lewis (2010) for monolingual selection and Axelrod et al. (2011) for bilingual selection, XenC uses cross-entropy (the average negative log of a sentence LM probabilities) as a metric to evaluate and sort those sentences.

Another motivation for our work on XenC is that a typical trend in SMT is to use as much data as possible to build statistical models, as long as this growing amount of data will provide a better BLEU score or any other translation quality automatic measure. However, the drawback of this trend is that the size of the models increases very quickly and become much more resource-demanding. So, in order to build either easily deployable systems or to estimate models on limited physical resources, it seems essential to consider resource usage like memory and computation time, both for models estimation and decoding process. Obviously, building a small system with very few data to attain this objective is quite trivial, but it often leads to important translation quality losses, so the goal of XenC is to provide a mean to extract small

¹<http://www.gnu.org/licenses/gpl.html>

amounts of data, carefully selected to match the desired translation task. This way, small but efficient systems can be built. Most of the time, performance of such systems will be better than a system built from all available but generic data, in terms of translation quality, memory usage and computation time.

3. Tool Description

XenC is a tool written in C++, which possesses four filtering modes. The common framework of all these modes is, from an *in-domain* corpus and one or several *out-of-domain* corpora, to first estimate two language models. Currently, all the LM estimations are handled by calls to the SRILM toolkit (Stolcke, 2002) libraries. These two models will then be used to compute two scores for each sentence of the *out-of-domain* corpus so the difference between these scores will provide an estimation of the closeness of each sentence regarding the considered task. In the remainder of this section, we will describe the modes and other functionalities proposed by XenC.

3.1. Processing Modes

The first mode is a filtering process based on a simple perplexity computation, as described in Gao et al. (2002). This is the simplest filtering mode proposed by XenC. Although it can provide interesting results and is less resource-demanding than the other modes, it is also less efficient.

The second mode is based on the monolingual cross-entropy difference as proposed by Moore and Lewis (2010). The cross-entropy is mathematically defined as:

$$H(P_{LM}) = -\frac{1}{n} \sum_{i=1}^n \log P_{LM}(w_i | w_1, \dots, w_{i-1}) \quad (1)$$

where P_{LM} is the probability of a LM for the word sequence W and w_1, \dots, w_{k-1} represents the history of the word w_i . In this mode, the first LM is estimated from the whole *in-domain* corpus. The second LM is estimated from a random subset of the *out-of-domain* corpus, with a number of tokens similar to the *in-domain* one. Formally, let I be our *in-domain* corpus and N our *out-of-domain* one. $H_I(s)$ will be the cross-entropy of a sentence s of N given by the LM estimated from I , while $H_N(s)$ will be the cross-entropy of sentence s of N given by the LM estimated from the subset of N . The sentences s_1, \dots, s_N from the *out-of-domain* corpus N will then be evaluated by $H_I(s) - H_N(s)$ and sorted by their score. Although this is a monolingual selection, this mode can be used efficiently on both monolingual and bilingual data.

The third mode is based on the bilingual cross-entropy difference as described in Axelrod et al. (2011). Unlike the second mode, we now take into account the two languages in our computations. Formally, let I_S and I_T be our *in-domain* corpus in source S and target T languages, and N_S and N_T our *out-of-domain* corpus with the same

language pair. For each language, we first compute the monolingual cross-entropy difference as described in the preceding paragraph. The final score will be computed by the sum between the two cross-entropy differences, as shown in the following equation:

$$[H_{I_S}(s_S) - H_{N_S}(s_S)] + [H_{I_T}(s_T) - H_{N_T}(s_T)] \quad (2)$$

where s_S is a word sequence from the *out-of-domain* corpus in source language and s_T is the corresponding word sequence from the *out-of-domain* corpus in target language.

The last mode operates similarly to the third one, but uses two phrase tables from the Moses toolkit (Koehn et al., 2007) as an input. Its goal is to adapt a phrase table considered as *out-of-domain* with another smaller phrase table considered as *in-domain*. First, source and target phrases are extracted from the phrase tables. Then, just like the third mode, LMs are estimated and used to score each *out-of-domain* phrase in each language. Finally, the scores are inserted in the original phrase table as a sixth feature. Another option is to compute local scores, relative to each unique source phrase. The redundant source phrases are merged into one structure containing their related target phrases, then the scores are computed locally and can be inserted in the original phrase table as a seventh feature. These two new features can then be added to the Moses configuration file for the *out-of-domain* translation system, and their weights tuned along with the other weights. Please note that this fourth mode is currently experimental and is barely tested.

3.2. Other Functionalities

Since the beginning of the XenC development right after the IWSLT 2011 evaluation campaign, back in December 2011, three main functionalities have been developed around the filtering modes to enhance them.

The first functionality added to XenC comes from an observation we made concerning the strong relation between the selected sentences and the random subset from the *out-of-domain* corpus. Indeed, the scores can vary significantly from one sample to another, impacting the resulting selection. Thus, we implemented a way to reduce this impact by optionally allowing to extract three random subsets instead of one for LM estimation. With this option, for each sentence to score, a cross-entropy measure is computed from each of the three language models. The three scores are then interpolated and used to compute the usual cross-entropy difference as described before. Our experiments shown that this option most of the time leads to a better selection than with only one random subset. It can be used within both the monolingual and bilingual cross-entropy filtering modes.

Our second added functionality is an option to perform the whole (monolingual or bilingual) filtering process on stemmed *in-domain* and *out-of-domain* corpora corresponding to the textual ones. These stemmed corpora must be created with an external tool. For this task, we recommend the TreeTagger tool (Schmid, 1995) which is efficient and language-independent. In order to ease the process of stemming the

corpora, a wrapper script exists within the Moses toolkit. Once the stemmed corpora are generated, distinct LMs and scores will be computed, then these scores will be merged with the ones from the original text corpora. Although this option is still experimental at the time of writing and has been barely tested, our initial experiments showed that an improvement can be achieved, and that integrating stems into the process can lead to a more heterogeneous selection, thus preventing the risk of increasing the number of out-of-vocabulary tokens (OOVs) in the resulting translation system. Again, this option is available for both the monolingual and bilingual filtering modes.

The third and last functionality implemented into XenC is the computation of cosine similarity measures in addition to the usual cross-entropy scores. In Information Retrieval, this measure is used for document clustering where each document is represented by a vector and vectors are compared by computing the cosine of the angle between them. By first determining a common vector of words, then considering the *in-domain* corpus as one document and each *out-of-domain* sentence as documents too, it is possible to obtain similarity scores for each sentence of the said corpus. Currently, XenC proposes two options regarding this similarity measure. It is possible to either combine this score with the cross-entropy one or to use it as a stand-alone selection criterion. Since this option has been added very recently, it is still highly experimental and needs extensive testing. To this date, no real improvements have been observed. Also, please note that this option is only available within the monolingual filtering mode.

Some other scoring options are available to fit different scoring needs. For instance, you can provide XenC a file containing weights for each sentence of the *out-of-domain* corpus. These weights can optionally be used as log values. Also, you can require a descending sorting order for your final scored file, which can prove useful when you need XenC to adapt to some existing scripts. Finally, by default, XenC proposes calibrated scores ranging from 0 (the best score) to 1 (the worst one). You can require our tool to invert those scores and have 1 being the best score and 0 the worst one.

4. Installation Requirements

In order to compile and install XenC from the source code right out-of-the-box, you will need a Linux (i386 or x86_64), Mac OSX (Darwin) or SunOS (Sparc or i386) operating system. Other platforms may work, but are totally untested. Also, you will need to dispose of the following third-party software:

- gcc version 4.2.1 or higher (older versions might work, but are untested),
- GNU make,
- gzip, to read/write compressed files,

- Boost² version 1.52.0 or higher (although it may work with lower versions, but is untested). XenC relies on the following multithreaded (“-mt” versions) libraries: filesystem, iostreams, program_options, regex, system and thread,
- SRILM³ version 1.7.0 or higher (older versions won’t work for sure, since they are not thread-safe). XenC relies on the following libraries: libdstruct, libmisc and liboolm.

Once all third-party software is installed, you can simply compile XenC by issuing the following command: `make`, or `make debug` if you want to keep the debug symbols. You can also specify custom paths for Boost or SRILM, by adding the `BOOST=` or `SRILM=` parameters.

5. Usage Instructions

By default, in order to run XenC, you need to provide at least:

- a source (and optionally target) language,
- an *in-domain* monolingual or parallel corpus,
- an *out-of-domain* monolingual or parallel corpus,
- a filtering mode.

The tool will then compute the *out-of-domain* sentences scores, generating all the needed vocabularies and language models when appropriate, and will output an ascending order sorted file (compressed with `gzip`), containing the scores in the first field and the sentences in the second (and third in case of parallel corpora) field(s). It is mandatory that the original corpora files do not contain tabulations. Empty lines are not an issue, since XenC will automatically skip them and also remove the corresponding sentences in the case of a parallel corpus. Automatic generation of needed files works as follows:

- for vocabularies, the words contained in the *in-domain* corpus will be used,
- for language models, estimation will be done using an order of four, a modified Kneser-Ney discounting and no cut-offs. LMs will be outputted in SRILM binary format.

You can of course provide your own vocabularies and LMs, and you can optionally change the order and the output format of the estimated LMs.

Concerning the evaluation process, it is based on perplexity computation of language models estimated from parts of various sizes of the sorted output file. Concretely, XenC will extract cumulative parts based on a fixed step size (usually ten percent), estimate language models on them, and then compute their perplexity against a development corpus. Our tool also propose a best point computation, which, from the evaluation mode perplexity distribution, will try to find the best percentage of the *out-of-domain* corpus to keep, based on a dichotomic search.

²<http://www.boost.org>

³<http://www.speech.sri.com/projects/srilm/download.html>

Regarding the performance, some parts of our tool are threaded, like the perplexity and cross-entropy computation (since the sentence order does not matter) as well as the language models estimation when evaluating. By default, XenC makes use of two threads, and we have successfully ran it with up to ten threads. But due to some memory leaks in the SRILM toolkit, the memory usage can become very important during the evaluation process. It is possible to limit this memory usage by requiring less threads, or by launching XenC twice, once for the selection process and once for the evaluation, instead of once for the whole procedure.

5.1. Usage Examples

The simplest command line which can be issued could be the following:

```
XenC -s fr -i indomain.fr -o outofdomain.fr -m 2 --mono
```

where `-s` indicates the source language, `-i` the *in-domain* corpus, `-o` the *out-of-domain* corpus, `-m` the filtering mode and `--mono` forces monolingual mode.

The following line:

```
XenC -s fr -i indomain.fr -o outofdomain.fr -m 2 --mono -e -d dev.fr
```

adds the evaluation mode (the `-e` switch) and `-d` provides the development corpus. To require best point computation, just replace the `-e` switch with the `-b` one.

The last example computes a bilingual filtering with a best point computation and eight threads:

```
XenC -s en -t fr -i indomain.en -o outofdomain.en -d dev.en \
--in-ttext indomain.fr --out-ttext outofdomain.fr -m 3 -b --threads 8
```

Please note that for now, the evaluation or best point can only be done on source language.

6. Experiments

We have performed a series of experiments based on the system we proposed for the IWSLT 2011 evaluation campaign, which achieved the first place in the speech translation task (Rousseau et al., 2011). This system was already based on a very basic perplexity data selection, which explain the fact that size reductions are not reported for the translation tables. We will present our results on selection for language modeling and translation modeling. For these selections, we consider the TED corpus as our *in-domain* one and all the other allowed corpora as our *out-of-domain* ones. The development and test corpora are the official sets proposed during the IWSLT 2010 campaign. Source language is English while target language is French. More detailed experiments can be found in Chapter 6 of Rousseau (2012).

6.1. Data Selection for Language Modeling

The original LM that we used for the evaluation campaign was estimated on all the available data, using a linear interpolation. To study the impact of the monolingual

Systems	dev2010 BLEU	tst2010 BLEU	LM size	
			On disk	In memory
IWSLT11 original	23.97	25.01	7.9G	22.1G
IWSLT11 XenC_LM	24.01	25.35	1.7G	5.2G

Table 1. BLEU scores and LM sizes with both original and reduced LMs.

Systems	dev2010	tst2010
IWSLT11 original	23.97	25.01
IWSLT11 XenC_monoEN	24.11	25.12
IWSLT11 XenC_monoFR	24.01	24.87
IWSLT11 XenC_biENFR	24.10	25.13

Table 2. BLEU scores for bilingual selection for translation models.

data selection, we performed it on each of the *out-of-domain* corpora and interpolated the resulting LMs linearly to create a new reduced LM. We ended up keeping only 11.3% of the original data according to the best point computation of XenC. Table 1 presents the BLEU scores obtained by our system for both the original LM and the reduced one, as well as the sizes of the two language models on disk and in memory. As we can observe, our reduced language model achieves better results than the original one, while requiring much less memory and disk space, thus also optimizing the decoding time and memory usage.

6.2. Data Selection for Translation Modeling

We also studied the impact of bilingual selection on all the *out-of-domain* corpora used for the translation model estimation. We made three different selections to compare the efficiency of bilingual selection to monolingual selection on both source and target sides. Table 2 shows the results obtained for each of these selections. As we can see, monolingual source selection and bilingual selection also achieve better results than the original system, while monolingual target selection reduce the translation quality and is therefore not suitable for translation models estimation.

6.3. Data Selection for the Whole System

After studying the individual impact of both monolingual and bilingual data selection, we combined the reduced models to observe if it is possible to achieve even better results than individual selections. Table 3 details the results obtained by the global systems for both monolingual source and bilingual selection. We can observe

Systems	dev2010	tst2010
IWSLT11 original	23.97	25.01
IWSLT11 XenC monoEN + LM	24.12	25.18
IWSLT11 XenC biENFR + LM	24.18	25.40

Table 3. BLEU scores for the complete experimental systems.

that although source monolingual and bilingual data selection results for the translation model were very similar when performed individually, we can achieve much better results with bilingual selection when the reduced language model is added to the system. In the end, we can report on this particular task a gain of 0.21 BLEU point on the development set and 0.39 BLEU point on the test set, which represents respectively a relative gain of 0.87% and 1.54%.

7. Conclusion and Perspectives

In this paper, we described XenC, an open-source tool for data selection in Natural Language Processing. While focusing our experiments on Statistical Machine Translation, we showed that with the help of our tool, carefully selecting the data injected in the building process of translation and language models dedicated to a specific task might lead to smaller models, reduced decoding time and better translation quality.

In the future, we plan to keep the tool development active, as we already have some improvements in mind:

- integrating other language model toolkits and particularly KenLM (Heafield, 2011) for speed and memory usage,
- proposing an option to use the full vocabulary of the two corpora, as it might lead to a reduced OOVs rate,
- extensively testing and enhancing the experimental functionalities,
- proposing an option to evaluate on the target language when doing bilingual selection.

Bibliography

- Axelrod, Amittai, Xiaodong He, and Jianfeng Gao. Domain adaptation via pseudo in-domain data selection. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 355–362, July 2011.
- Gao, Jianfeng, Joshua T. Goodman, Mingjing Li, and Kai-Fu Lee. Toward a unified approach to statistical language modeling for Chinese. In *ACM Transactions on Asian Language Information Processing (TALIP)*, volume 1, pages 3–33, March 2002.
- Heafield, Kenneth. KenLM: faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, July 2011.

- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Meeting of the Association for Computational Linguistics*, pages 177–180, 2007.
- Moore, Robert C. and William Lewis. Intelligent selection of language model training data. In *Proceedings of the ACL Conference Short Papers*, pages 220–224, July 2010.
- Rousseau, Anthony. *La Traduction Automatique De La Parole*. PhD thesis, Université du Maine, December 2012.
- Rousseau, Anthony, Fethi Bougares, Paul Deléglise, Holger Schwenk, and Yannick Estève. LIUM’s systems for the IWSLT 2011 speech translation tasks. In *Proceedings of International Workshop on Spoken Language Translation*, pages 79–85, December 2011.
- Schmid, Helmut. Improvements in part-of-speech tagging with an application to German. In *Proceedings of the ACL SIGDAT-Workshop*, pages 47–50, 1995.
- Stolcke, Andreas. SRILM – an extensible language modeling toolkit. In *Proceedings of Inter-speech*, pages 901–904, September 2002.

Address for correspondence:

Anthony Rousseau

`anthony.rousseau@lium.univ-lemans.fr`

Laboratoire d’Informatique de l’Université du Maine (LIUM)

Avenue Laënnec

72085 LE MANS CEDEX 9, France



**COSTA MT Evaluation Tool:
An Open Toolkit for Human Machine Translation Evaluation**

Konstantinos Chatzitheodorou^a, Stamatis Chatzistamatis^b

^a Aristotle University of Thessaloniki
^b Hellenic Open University

Abstract

A hotly debated topic in machine translation is human evaluation. On the one hand, it is extremely costly and time consuming; on the other, it is an important and unfortunately inevitable part of any system. This paper describes COSTA MT Evaluation Tool, an open stand-alone tool for human machine translation evaluation. It is a Java program that can be used to manually evaluate the quality of the machine translation output. It is simple in use, designed to allow machine translation potential users and developers to analyze their systems using a friendly environment. It enables the ranking of the quality of machine translation output segment-by-segment for a particular language pair. The benefits of this tool are multiple. Firstly, it is a rich repository of commonly used industry criteria (fluency, adequacy and translation error classification). Secondly, it is freely available to anyone and provides results that can be further analyzed. Thirdly, it estimates the time needed for each evaluated sentence. Finally, it gives suggestions about the fuzzy matching of the candidate translations.

1. Introduction

Machine translation (MT) refers to the use of a machine for performing translation tasks which convert a text from a source language into a target language. Given that there may exist more than one correct translation of any given sentence manual evaluation of MT output is difficult and persistent problem. On the one hand, it is “holy grail” in MT community; on the other, it is becoming impractical because it is a time-consuming, costly and, sometimes, a subjective process. Answering questions about the accuracy and fluency, and categorizing translation errors are just as important as

the MT itself. Moreover, human evaluation results give the opportunity to compare system performance and rate its progress. At the same time, researchers suffer from the lack of suitable, consistent, and easy-to-use evaluation tools.

During the DARPA GALE evaluations (Olive et al., 2011), a similar tool was designed but it was only made available to participants in the GALE program. Appraise is an other open-source tool for manual evaluation of MT output. It allows to collect human judgments on translation output, implementing annotation tasks such as translation quality checking, ranking of translations, error classification, and manual post-editing. It is used in the ACL WMT evaluation campaign (Federmann, 2012). Last but not least, PET is a stand-alone tool that has two main purposes: facilitate the post-editing of translations from any MT system so that they reach publishable quality and collect sentence-level information from the post-editing process, e.g.: post-editing time and detailed keystroke statistics (Aziz et al., 2012).

We implemented a simple stand-alone tool which facilitate MT evaluation as much as possible and to give easy access to collected evaluation data for further analysis. The typical requirements of such a tool in the framework of machine translation (MT) research are discussed in this section. Section 2 discusses usage and the corresponding graphical user interface of the tool as well the analysis of the results. Section 3 describes the evaluation criteria used and, finally, Section 4 concludes and gives an outlook on future work.

2. The Tool

COSTA MT Evaluation Tool helps users to manually evaluate the quality of the MT output. The tool uses standard Java libraries; hence it works on any platform running a Java Virtual Machine. There is no special installation; the tool runs by just double clicking the file into any target directory.

2.1. Usage

Each evaluation task in COSTA MT Evaluation Tool is called a “project”. Each project requires the user to provide three parallel text files (UTF-8). Every line of these files should contain one sentence.

1. Source file contains the source sentences.
2. MT file contains the candidate translations.
3. Reference file contains the reference translations.

COSTA MT Evaluation Tool gives the opportunity to the user to choose the number of sentences and interrupt or restart the project at any time. Moreover, users can have many projects on hold. The main window of the tool is divided into 4 parts: i) the part of the source text, ii) the part of the machine translation, iii) the part of the reference translation, and iv) the part of the translation error classification as shown in Figure 1.

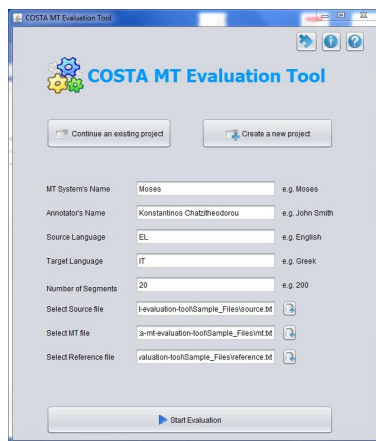


Figure 1. Main screen

By pressing NEXT, a new sentence comes for evaluation if the current sentence is already evaluated. Annotators can stop the evaluation process by pressing Stop & Get at any time. In that case, the results for all the already evaluated sentences will be counted.

2.2. Getting Results

COSTA MT Evaluation Tool presents the users with automated reports on the results of evaluations as it is shown in Figure 3. Once evaluation is completed, the Tool will create a text file (UTF-8) in the target directory. The base filename consists of <the system's name> + <the annotator's name> + <the source> and <target> languages, followed by _results.txt. For instance, a typical name for a Moses English into Greek MT system with annotator Mr. Smith will be:

Moses_Smith_EN_GR_results.txt

This file can easily be imported into Excel, SPSS, MATLAB, and most other statistical software suites for further analysis and significance testing. In addition, it can be read by other tools or machine learning algorithms in order to estimate the quality of future MT outputs. The header of the file contains all the information for the system as well as the average fluency and adequacy scores and the count of the errors. Moreover, each line of the rest of this file contains the analytical results for each evaluated sentence.

COSTA MT Evaluation Tool

COSTA MT Evaluation Tool

Stop & get results Help

Source: Deputati per Stato membro e gruppo politico

MT: MEP by Member State and political group

Fluency: ☐ 1. Incomprehensible ☒ 2. Disfluent language ☐ 3. Non-native language ☐ 4. Good language ☐ 5. Flawless language

Reference: MEPs by Member State and political group

Accuracy: ☐ 1. None ☐ 2. Little meaning ☐ 3. Much meaning ☒ 4. Most meaning ☐ 5. All meaning

Sentence: 1/10 Next

Translation error classification

Grammar: ☐ Verb inflection ☐ Noun inflection ☐ Other inflection ☐ Wrong category ☐ Article ☒ Preposition ☐ Agreement

Comments: pe

Words: ☐ Single words ☐ Multi-word units ☐ Terminology ☒ Untranslated words ☐ Ambiguous translation ☐ Literal translation ☐ Conjunctions


Comments:

Style: ☐ Acronyms - Abbreviations ☐ Extra words ☐ Country standards ☐ Spelling errors ☐ Accent ☐ Capitalization ☐ Punctuation

Comments:

Figure 2. Evaluation environment

COSTA MT Evaluation Tool



COSTA MT Evaluation Tool

Download results

Fluency: 0.6 / 1 1 sec average time per sentence

Adequacy: 0.74 / 1 0 sec average time per sentence

Fluency		Adequacy:	
Incomprehensible	1	None	1
Good language	4	Little meaning	1
Non-native language	0	Much meaning	2
Disfluent language	4	Most meaning	2
Flawless language	1	All meaning	4

Grammar	Words	Style
Verb inflection	Single words	Acronyms - Abbreviations
Noun inflection	Multi-word units - Idioms	Country standards
Other inflection	Terminology	Spelling errors
Wrong category	Untranslated words	Accent
Article	Incomprehensible	Capitalization
Preposition	Literal translation	Punctuation
Agreement	Conjunctions	

Figure 3. Evaluation results

3. Evaluation Metrics

COSTA MT Evaluation Tool enables users to evaluate the MT performance using the two main criteria:

1. Fluency and adequacy
2. Translation error classification

3.1. Fluency and Adequacy

The objective of the fluency evaluation is to determine how “fluent” a translation appears to be, without taking into account the correctness of the information. The evaluation does this segment-by-segment on a 1–5 scale without referring to any reference text. The objective of the adequacy evaluation is to determine the extent to which all of the content of a text is conveyed, regardless of the quality of the language in the candidate translation. The evaluation does this segment-by-segment on a 1–5 scale. The annotator is given the following definitions of adequacy and fluency (Koehn, 2007):

Fluency	Adequacy
5. Flawless language	5. All meaning
4. Good language	4. Most meaning
3. Non-native language	3. Much meaning
2. Disfluent language	2. Little meaning
1. Incomprehensible	1. None

Since, recent evaluation campaigns have shown that judgments of fluency and adequacy are closely related, COSTA MT Evaluation Tool firstly asks annotators to evaluate the fluency without referring to any reference text and secondly the adequacy with reference to the reference text (White, 1995). The evaluation of translation error classification is optional.

3.2. Translation Error Classification

During the evaluation of fluency and adequacy, COSTA MT Evaluation Tool offers users the option to count and categorize errors. This type of evaluation can provide a descriptive framework that reveals relationships between errors. Furthermore, it can also help the evaluator to map the extent of the effect in chains of errors, allowing comparison among MT systems. At the same time, we propose these criteria as a new methodology of human translation error classification.

In total, there are three main categories each with seven subclasses. These categories were identified by observing the most frequent error types in MT outputs among Moses-based (Koehn et al., 2007) and free MT systems such as Google Translate and Bing Translator.

The first category concerns the grammatical and the linguistic accuracy of the machine translated texts. The second category concerns the use of the vocabulary and the third the format and style of the produced texts. Analytically, translation error classification works to the following criteria:

Linguistic

Verb inflection	Incorrectly formed verb, or wrong tense.
Noun inflection	Incorrectly formed noun (e.g. as nominative nouns in apposition).
Other inflection	Incorrectly formed adjective or adverb.
Wrong category	Category error (e.g. noun vs. verb).
Article	Absent or unneeded article. (e.g. The London vs. London)
Preposition	Incorrect, absent or unneeded preposition.
Agreement	Incorrect agreement between subject-verb, noun-adjective, past participle agreement with preceding direct object, etc.

Words

Single words	Sentence elements ordered incorrectly.
Multi-word units	Incorrect translation of multi-word expressions and idioms (e.g. to pay a visit).
Terminology	Incorrect terminology.
Untranslated words	Word not in dictionary.
Ambiguous translation	Ambiguous target language.
Literal translation	Word-for-word translation.
Conjunctions	Failure to reconstruct parallel constituents after conjunction, or failure to identify boundaries of conjoined units.

Style

Acronyms – Abbreviations	Incorrect abbreviations, acronyms and symbols.
Extra words	Extra words in target language.
Country standards	Incorrect format of dates, addresses, currency etc.
Spelling errors	Misspelled words.
Accent	Incorrect accents.
Capitalization	Incorrect upper or lower case.
Punctuation	Punctuation is incorrect, absent or unneeded.

There are also three additional boxes to the bottom of the main screen, one for each translation error category, and where the evaluator could add comments.

4. Conclusion and Future Work

We have presented a simple tool for manual evaluation of MT. It is simple in use, designed to allow potential MT users and developers to analyze their systems using a friendly environment. It enables the ranking of the quality of MT output segment-by-segment for a particular language pair. At the same time, we propose these criteria as a new methodology of human translation error classification. Our future work includes:

1. Multiple MT systems evaluation
2. Multiple Reference evaluation
3. Extraction of feature that can be analyzed by machine learning algorithms for the estimation of the MT quality without reference translation.

The tool is available for download at:

<https://code.google.com/p/costa-mt-evaluation-tool/>

Bibliography

- Aziz, Wilker, Sheila Castilho Monteiro de Sousa, and Lucia Specia. PET: a tool for post-editing and assessing machine translation. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA). ISBN 978-2-9517408-7-7.
- Federmann, Christian. Appraise: An open-source toolkit for manual evaluation of machine translation output. *The Prague Bulletin of Mathematical Linguistics*, 98:25–35, September 2012.
- Koehn, Philipp. *Statistical Machine Translation*. Cambridge University Press, 2007.
- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.
- Olive, Joseph, Caitlin Christianson, and John McCary. Handbook of natural language processing and machine translation: DARPA global autonomous language exploitation. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. Springer, 2011.
- White, John S. Approaches to black box MT evaluation. In *MT Summit V Proceedings*, July 1995.

Address for correspondence:

Konstantinos Chatzitheodorou

chatzik@itl.auth.gr

Aristotle University of Thessaloniki

University Campus, GR-54124 Thessaloniki, Greece



The Prague Bulletin of Mathematical Linguistics

NUMBER 100 OCTOBER 2013 91-100

Open Machine Translation Core: An Open API for Machine Translation Systems

Ian Johnson

Capita Translation and Interpreting

Abstract

Open Machine Translation Core (OMTC) is a proposed open API that defines an application programming interface (API) for machine translation (MT) systems. The API defined is a service interface which can be used to underpin any type of MT application. It consists of components which allow programmers, with little effort, to integrate different MT back-ends into their applications since an OMTC compliant MT system presents a consistent interface. OMTC attempts to standardise the following aspects of an MT system: **resources** – the abstract representation of assets used e.g. documents and translation memories, **sessions** – a period of time in which a user interacts with the system, **session negotiation** – agreement on which services are to be provided, **authorisation** – integration with third party authorisation systems to prevent users performing unauthorised actions, **scheduling** – the management of long running MT tasks, **machine translation engines** – a representation of an entity capable of providing only MT, and **translators** – a conglomeration of, at least one of the following, an MT engine, a collection of translation memories, and a collection of glossaries.

1. Introduction

Open Machine Translation Core (OMTC) is a proposed and open API for the construction of machine translation (MT) systems (Johnson, 2013). The central idea of OMTC is to be able to easily integrate disparate back-end MT systems together into an application such that the back-ends “look” consistent no matter the flavour of MT.

To identify the aspects and concerns that would be common to MT systems a use case analysis was carried out. Once the *actors* and use cases were catalogued then use cases which *any* MT system would require were identified. This reduced set was expanded into UML class diagrams to define the abstract OMTC specification. How-

ever, OMTC does define concrete classes where necessary. This paper gives a fairly high level description of the OMTC specification with a view that the reader study the full specification for details. OMTC attempts to standardise:

- **Resources:** the abstract representation of assets used by users in an MT system, e.g. documents and translation memories,
- **Sessions:** a period of time in which a user interacts with the system, e.g the time between login and logout,
- **Session Negotiation:** agreement on which services are to be provided,
- **Authorisation:** integration with third party authorisation systems to prevent users performing unauthorised actions,
- **Scheduling:** the management of long running and computationally expensive MT tasks,
- **Machine Translation Engines:** a representation of an entity capable of providing only MT, and
- **Translators:** a conglomeration of, at least one of the following, an MT engine, a collection of translation memories, and a collection of glossaries.

Figure 1 shows an example of how OMTC could be implemented. The figure shows two example applications: a client-server and a command line application. OMTC sits low down in the stack. OMTC's position gives the application programmer much more flexibility and freedom to use technologies and networking protocols that are available to them. For example, TAUS published their open MT system API which is designed to work as a RESTful web-service over HTTP (TAUS, 2012). Implementers of this API are tied to using HTTP. Using HTTP may not be desirable in some customer deployments, for example messages queues may have to be used. OMTC, on the other hand, is not tied to any technology and is reusable since it concentrates on one aspect of an MT system: *machine translation*. Moreover, the TAUS API specifies which methods are available to consumers of their service. If methods or arguments are required to be augmented the implemented MT system becomes non-compliant. OMTC allows the implementer to specify the methods and arguments required for their MT system.

Below OMTC sit the translation providers. Figure 1 shows the following disparate MT systems:

- **SmartMATE:** a self-serve SMT system allows API calls, via its RESTful web-interface, to build translation engines and start translations (Way et al., 2011).
- **Moses:** an open-source suite of tools for SMT engine development and translation. Integrating to an OMTC system would probably take the form of wrapping the existing command-line tools (Koehn et al., 2007).
- **SYSTRAN:** A rule-base MT system with an API available in their *Enterprise Server* product (SYSTRAN, 2008).
- **SDL Trados:** A computer-aided translation suite which presents an API called *SDL OpenExchange* (<http://www.sdl.com/products/sdl-trados-studio/>).

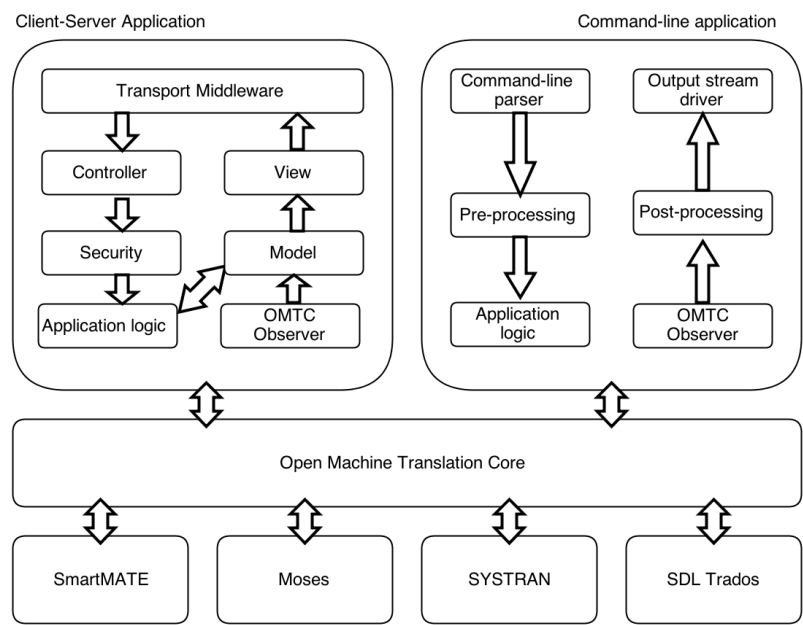


Figure 1. Example OMTC compliant applications

OMTC attempts to make these proprietary APIs homogeneous by defining an abstract interface for machine translation tasks and maintenance.

Further to the abstract specification, a reference implementation has been constructed using Java v1.7. It is released under a LGPL v3 license and is available by cloning the GitHub repository <https://github.com/ianj-als/omtc.git>. This implementation was written to provide an implementation that could be immediately used by developers to write OMTC compliant applications. The OMTC reference implementation is being used, at Capita Translation and Interpreting, to re-factor the SmartMATE application.

There follows a brief description of the common *actors* that would use an MT system. These *actors* were the central basis on which OMTC was designed.

2. Actors

An actor specifies a role played by a “user” that interacts with a system, but is not a part of that system. Actors are external to the system and can represent a human, external software, or an external system etc. (Obj, 2007).

There are three principal actors in an MT system:

- **Translator:** This actor's role is to perform translations and is the main end-user of an MT service. All other actors provide means to provide resources so that the translator may schedule translation tasks. Since this actor is expected to be widespread it attracts the fewest number of possible actions in the MT service and those actions are primarily read-only. Therefore, the scope to which this actor can intentionally harm the MT service is kept to a minimum. Moreover, this actor requires very little knowledge of MT in order to complete translation tasks.
- **Engine Manager:** This actor is able to mutate MT engines. The primary role of this actor is to maintain MT engines, e.g., train, re-train, compose or destroy MT engines. This actor should have a reasonable understanding of MT and the kinds of MT that the implementation is supporting. For example, if the implementation supports SMT then this actor would have an understanding of how to take a tabula rasa system and build an engine for use by the translator actor. Moreover, this actor is responsible for determining who is able to use the engines which the actor constructs. The use cases available to the engine manager actor is the union of those use cases for this and the translator actor.
- **Administrator:** The administrator actor is permitted to manage users for a particular customer. Customers may have many users which are translators or engine managers. Managing which use cases a customer's users are permitted to perform is the administrator actor's remit. This actor would be authorised to choose the payment plan, if one is required, and make payments for the use of the MT service. The administrator actor is permitted to invoke the use cases available to engine manager and translator actors.

Considering each of these actors, a number of *concerns* have been arrived at that are believed to be common to many MT systems. The concerns are collections of use cases and are described below.

3. Resource Management

A *resource* is an object that is provided or constructed by a user action for use in an MT system. A non-exhaustive list of examples is: document files, translation memories, glossaries, or MT engines. *Resource management* is a collection of use cases that allow all actors to load, construct, catalogue, and remove resources from an MT system. For example, if the MT system were a web-service then making a translation memory available to the MT system would probably be an upload action.

Resources may need some kind of ownership. If an MT system is a standalone command line driven application this may not be necessary, or the file-system can provide this feature: read or write permissions on files being used by the running process will be determined by the user running the process. However, if an MT system is a multi-user service then ownership of resources would become necessary. Users

from different customers should not be permitted to access *any* resource constructed or made available to the system by other customer users.

OMTC defines two kinds of resource:

1. **Primary resources:** any resource that has been constructed externally and made available, in some way, for use in an MT system. Examples of these resources are: a document, a translation memory (TM), a glossary etc. If these resources are required for future use it is recommended that these resources be persisted. Primary resources are immutable, i.e. if a resource's content is to be altered it is a distinct resource.
2. **Derived resources:** these resources are constructed using their primary counterparts either as a conglomeration or a separate entity is created, e.g. creating a SMT engine using a translation memory (a primary resource) to create a derived resource: the engine itself.

4. Sessions, Negotiation and Authorisation

In order for users to be able to use an MT service the API needs an idea of a *session*. A session is the period in which a user will interact with an MT service. An MT application may need to acquire the identity of users, whilst other implementations may not. Therefore, the OMTC API needs to support both user identity and anonymity. Moreover, clients to an MT service will support certain exchange formats, and expect certain features from the application. A *session negotiation* is defined in the API in order that both client and server can ascertain if, once the session is set up, their expectations of each other is correct. If a user's identity is to be determined then the application can restrict the actions a user can perform based on their role(s), i.e. *authorisation*. OMTC models these aspects.

4.1. Sessions

A session is a period in which a user interacts with an MT system. OMTC places no restrictions an application's definition on a session other than this. Sessions could be defined by the time between login and logout, the lifetime of a console application, or persisted over many login/logouts. Sessions can be associated with a user where user identity is necessary, for example in a pay-as-you-go web-application. In applications where user identity is not required an OMTC session supports not being associated with a user. An example of this type of application would be a console command line application where the user is explicit: the user running the program. All actions in an OMTC application are done on behave of a session.

4.2. Session Negotiation

An optional part of the OMTC specification is *session negotiation*. Session negotiation is a protocol which allows the provider and consumer of an MT service to come

to some agreement on what can be expected from the provider. If session negotiation is implemented clients, including other MT systems, can discover which features are supported, and which requirements are necessary. The features and requirements are modelled as *capabilities*. Capabilities come in four flavours:

- **API:** This capability, today, only specifies the version of the API being used.
- **Resources:** These capabilities describe the file types that the service can support. Supporting means that the service will store and use the resource in an appropriate way.
- **Features:** The actions that can be expected from an MT service, but may not be available in every MT service.
- **Prerequisites:** The prerequisites that client shall ensure are true before some or all of the MT service's features become unavailable to a client, e.g. payment.

During negotiation the unsupported capabilities are returned to the consumer. If provider has determined that the consumer cannot have a meaningful conversation then the session is closed. However, the consumer can close the session if it receives unsupported capabilities on which it depends. Session negotiation must be completed before the consumer completes session initialisation.

4.3. Authorisation

OMTC does not specify *any* security features. It is the application's responsibility to integrate with authentication systems. However, if authorisation is required in an MT system then some integration with the external authentication provider is necessary to provide user identity, and authorisations. The specification provides two interfaces to interlock an external authentication provider.

5. Scheduling

Machine translation consists of a number of operations which are computationally expensive. Constructing an MT service with many users requires that the computational resources are shared *fairly* between the demands of the users. The implementer of an MT service needs to define:

- Which computational resource or resources will be used to execute the computationally expensive operations,
- The latency of an operation before it is executed, and
- A policy to determine how users' operations will be scheduled, i.e. priority.

The scheduling API, defined by OMTC, needs to support different kinds of computation resource management: from native threading to distributed resource management products. The pattern used in the scheduling API is detached execution with notification on completion, whether successful or not.

5.1. Tickets

The scheduling API issues *tickets* when an operation is submitted to the underlying detached execution implementation. A ticket is a receipt for, and uniquely identifies an operation. When the operation is submitted an *observer* will be provided which observes the progress of the computation. On completion, the observer is invoked with the appropriate ticket to identify which operation has completed. This is the observer design pattern (see Gamma et al., 1994). The observer is application defined and is used to update any data that relies on the computation.

Operation priorities are defined using the scheduling API. This allows an application defined priority to be used to prioritise operations into the particular detached execution environment. For example, a priority could, say, for a paid-for MT service prioritise operations, invoked by users, which are on a higher tariff. So, say, a user on a *Freemium* tariff would have their operations prioritised lower than a user who pays for the service. Depending on the detached execution environment a priority might determine, not only, the latency of an operation, but also how much processor time a certain operation can expect when being execute.

6. Machine Translation Engines

A *machine translation engine* is defined as an entity that will solely perform machine translation. This may be a decoding pipeline in an SMT system or software that implements a rule based system. MT engines are built using primary resources and generally use computationally expensive operations to produce translations. Engines shall have operations available that, depending on their nature, shall read or mutate engine state, e.g.

- Evaluating an engine,
- Composing engines,
- Testing engines, and
- Training SMT engines.

Mixin interfaces are used to add optional functionality to an MT engine. This allows the application programmer to choose mixins useful to the kind of MT engine being implemented. Using mixins in this way prevents the application programmer from being tied to this API; it does not mandate that any class inheritance is used. This is particularly useful when using languages that do not support multiple inheritance and can be used alongside existing frameworks and class hierarchies.

The mixins provided define the following operations:

- **Composition:** compose one MT engine with another,
- **Evaluation:** score an MT engine,
- **Update parameters:** mutate runtime options/parameters,
- **Querying:** invoking translations one sentence at a time,

- **Training and retraining:** specifically for SMT engines to build appropriate models,
- **Testing:** provide resources to test a constructed MT engine, and
- **Updating:** mutation of an existing engine to adapt to new data or rules.

The operations, in the mixins, that could represent computationally expensive operations and use an asynchronous invocation pattern. In order to track the operation the caller of these methods receives a *ticket*. The ticket is used to represent an “in flight” operation and, once complete, will be used in a notification. Notifications are used to inform the application of the state of a completed operation: submitted, starting, or completed successfully or failed.

7. Translators

Translators are a conglomeration of an MT engine, translation memories and glossaries. A translator will specify at least one of these resources. This allows translators to support translations using any combination of MT, TM or glossaries. It is the responsibility of application programmers to handle these resources in an appropriate way for the flavour of translation required.

Translations are typically computationally expensive and can take a considerable amount of time to complete. In an MT system that is multi-user computation resources should be shared fairly between the demands of the submitted translations. As with MT engine operations, translations shall be ticketed and a ticket observer is required to receive notifications of the progress of a translation task.

There are two methods of performing a translation:

- **Primary Resource Translation:** A primary resource made available to an MT system can be translated. It is application defined as to which kind of primary resources are supported for translation. If supported, it is implementation defined as to whether any pre- or post-processing is required, e.g. file filtering.
- **Sentence-by-sentence Translation:** Translations can be supported that consist of a single sentence. MT engines can be queried sentence-by-sentence to perform a translation using only the engine. However here, TM and glossaries can be mixed into a richer translation that uses any translation pipeline that may be implemented.

8. Language Bindings

The OMTC specification is documented using UML which is a language-agnostic representation. It is expected that any modern computing language is capable of implementing the OMTC specification. OMTC defines some *generalised classes*. Generalised classes are classes which require types arguments to construct the class. Concrete implementations of this is are Java and C# generics, and C++ templates. Many, if not all, of the extant non-object oriented computing languages are not capable of im-

plementing these classes. However, the solution is to design an OMTc implementation that builds concrete representations of the OMTc generalised classes. Functional programming languages are also candidates for use in implementations. Haskell's typeclass language feature would be particularly suited to an OMTc implementation.

The OMTc specification comes with a Java v1.7 reference implementation. This implementation was constructed to allow people to view the specification in code to gain deeper understanding, and, if they wish, to build their own OMTc compliant MT system with Java.

Implementations in other languages are encouraged. With the popularity of web-frameworks, such as Spring MVC (Yates et al., 2013), Rails (Hart, 2012) and Django (Alchin, 2013), Ruby and Python implementations are welcome since they'll provide an easy way to build web-hosted MT services.

9. Summary

A proposed open API for MT systems, called Open Machine Translation Core, has been presented. It attempts to standardise common aspects and concerns to all MT systems. It is believed that this abstract interface will underpin and ease the development of *any* MT system being developed. Whilst this is only a high level view of the proposed API it is recommended that the reader view the full and entire specification. The full specification and a Java reference implementation is freely available, under a LGPL v3 license, from GitHub by cloning <https://github.com/ianj-als/omtc.git>.

Acknowledgements

This work was done as part of the MosesCore project sponsored by the European Commission's Seventh Framework Programme (Grant Number 288487).

Bibliography

Alchin, Marty. *Pro Django*. Apress, 2nd edition, 2013.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1st edition, 1994.

Hart, Michael. *Ruby on Rails Tutorial: Learn Web Development with Rails*. Addison Wesley, 2nd edition, 2012.

Johnson, Ian. *OMTC: Open Machine Translation Core*, Version 0.6.1-DRAFT edition, 2013. URL <https://github.com/ianj-als/omtc/blob/master/documentation/omtc.v0.6.1-DRAFT.pdf>.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical

- machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, ACL '07*, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
- OMG *Unified Modeling Language (OMG UML), Superstructure, V2.1.2*. Object Management Group, Inc., 2007. URL <http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF>.
- SYSTRAN (2008). *SYSTRAN Enterprise Server 6: API Reference Guide*. SYSTRAN, 2008. URL <http://www.systransoft.com/download/user-guides/SYSTRAN.ses6-api-reference-guide.pdf>.
- TAUS(2012). *A Common Translation Services API*. TAUS, September 2012. URL <https://labs.taus.net/interoperability/taus-translation-api>.
- Way, Andy, Kenny Holden, Lee Ball, and Gavin Wheeldon. SmartMATE: Online self-serve access to state-of-the-art SMT. In *Proceedings of the Third Joint EM+/CNGL Workshop “Bringing MT to the User: Research Meets Translators”*, pages 43–52, 2011.
- Yates, Colin, Seth Ladd, Marten Deinum, Koen Serneels, and Christophe Vanfleteren. *Pro Spring MVC: With Web Flow*. Apress, 2nd edition, 2013.

Address for correspondence:

Ian Johnson
ian.johnson@capita-ti.com
Capita Translation and Interpreting
Riverside Court, Huddersfield Road
Delph, Lancashire
OL3 5FZ, United Kingdom



The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013 101-112

**CASMACAT: An Open Source Workbench
for Advanced Computer Aided Translation**

Vicent Alabau^a, Ragnar Bonk^b, Christian Buck^c, Michael Carl^b,
Francisco Casacuberta^a, Mercedes García-Martínez^b, Jesús González^a,
Philipp Koehn^c, Luis Leiva^a, Bartolomé Mesa-Lao^b, Daniel Ortiz^a,
Herve Saint-Amand^c, Germán Sanchis^a, Chara Tsoukala^c

^a Institut Tecnològic d'Informàtica, Universitat Politècnica de València, Spain

^b Copenhagen Business School, Department of International Business Communication, Denmark

^c School of Informatics, University of Edinburgh, Scotland

Abstract

We describe an open source workbench that offers advanced computer aided translation (CAT) functionality: post-editing machine translation (MT), interactive translation prediction (ITP), visualization of word alignment, extensive logging with replay mode, integration with eye trackers and e-pen.

1. Introduction

The use of machine translation technology among professional human translators is taking hold rapidly, but there is only very limited research on this man-machine collaboration, especially compared to the vast current research push on core machine translation technology. We believe that big part of this reason is that there are no sufficient open source platforms that lower the barrier to entry.

To resolve this, two EU-funded research projects, *CASMACAT*¹ and *MATECAT*,² are committed to develop an open source workbench targeted both at researchers to investigate novel and enhanced types of assistance and at professional translators for actual use. Through this combined effort, we hope to kick-start broader research into computer aided translation methods, facilitating diverse translation process studies, and reach volunteer and professional translators without advanced technical skills. At the mid-point of the 3-year projects, we release this tool as open source software. In this paper, we focus on *CASMACAT*'s contributions and give instructions for installation and use of the workbench.

2. Related Work

A number of academic studies have shown that post-editing machine translation can be more efficient than translation from scratch (Plitt and Masselot, 2010; Skadiņš et al., 2011; Pouliquen et al., 2011; Federico et al., 2012), as is also evident from recent trends in industry adoption. But post-editing machine translation is not the only approach. The idea of so-called interactive machine translation was pioneered by the *TRANSType* project (Langlais et al., 2000) and has been further developed in the following decade (Barrachina et al., 2009; Koehn, 2010).

We are not aware of any fully featured open source tool for computer aided translation research. A related open source project is *OMEGAT*, an editor with translation memory system, written in Java, targeted at freelance translators. We will explore integration of the functionalities of the *CASMACAT* workbench into this tool in the future.

3. Usage

The *CASMACAT* UI consists of views designated for different tasks. The translate view is its central view, where the user can translate a document and post-editing assistance and logging takes place. Other views offer a way to upload new documents or to manage the documents that are already in the system. Also, a replay mode has been implemented. The different views will now be shown and described in the sequence they are typically used.

3.1. Upload

If the user opens the default URL without giving any special parameters, she is taken to the upload view. This is currently the entry point of the application. At this point a user can specify one or several documents to upload and to translate. The documents uploaded must be in XLIFF format. The language pair can either be chosen

¹<http://www.casmacat.eu/>

²<http://www.matecat.com/>

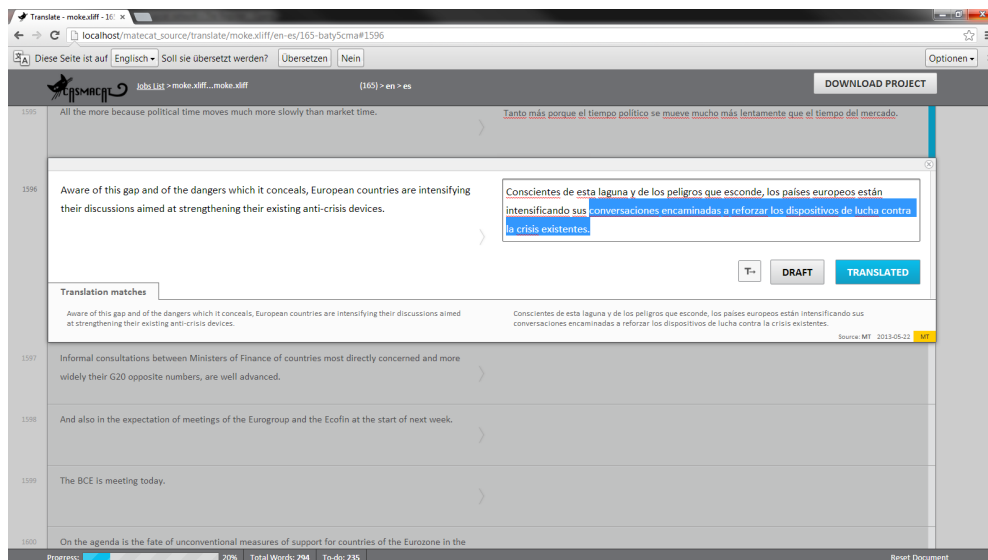


Figure 1. Translate view with post-editing configuration

manually or auto-detected from the XLIFF file. If several documents are uploaded at once, they are bundled into one job and are translated in a sequence. If the user clicks on the *Start Translating* button she is taken to the translate view and can start working.

3.2. Editing

In the translate view, the user can now translate the document (see Figure 1). The document is presented in segments, while the currently active segment is highlighted and assistance is provided for this segment. If using the post-editing configuration without ITP up to three MT or TM suggestions are provided, from which the user can choose. The user can use shortcuts, for instance, to go to the next segment or to copy the source text to the target. The user can assign different states to a segment, for instance, *translated* for finished ones or *draft* for segments, where she is not yet sure about the translation and she wants to review later. When finished, the *Download Project* button may be used to download the translated document, again in the XLIFF format.

A screenshot of a text editor window. The text inside is "Lisboa y Madrid quieren emprender un camino diferente del adoptado por Grecia e Irlanda." The word "emprender" is highlighted in blue, and a small box is open above it, showing the word "camino" in red, which is the prediction for the next word in the sentence.

Lisboa y Madrid quieren emprender un camino diferente del
adoptado por Grecia e Irlanda.

Figure 2. Interactive Translation Prediction

4. Features

In this section we present a short description of the main advanced CAT features that we implemented in the workbench. The common goal of these features is to boost translator productivity.

4.1. Post-Editing Machine Translation

The default mode of the workbench is post-editing of either machine translation output or of matches from translation memory systems. This mode of operation is the minimal deviation from traditional work practice of professional translators, and hence the most conservative type of assistance offered.

4.2. Intelligent Autocompletion

The main alternative is interactive translation prediction, where new machine translation predictions are generated every time a keystroke is detected by the system (Barachina et al., 2009). In such event, the system produces a prediction for the rest of the sentence according to the text that the user has already entered. This prediction is placed at the right of the text cursor.

Providing the user with a new prediction whenever a key is pressed has been proved to be cognitively demanding (Alabau et al., 2012). Therefore, we decided to limit the number of predicted words that are shown to the user by only predicting up to the first erroneous word according to confidence measures.

In our implementation, pressing the Tab key allows the user to ask the system for the next set of predicted words. See Figure 2 for a screenshot.

4.3. Confidence Measures

Confidence measures inform the user about which part of the translation is more likely to be wrong than others (González-Rubio et al., 2010). We use confidence measures under two different criteria. On the one hand, we highlight in red color those translated words that are likely to be incorrect. We use a threshold that favors precision in detecting incorrect words. On the other hand, we highlight in orange color those translated words that are dubious for the system. In this case, we use a threshold that favors recall.

4.4. Search and Replace

Most of the computer-assisted translation tools provide the user with intelligent search and replace functions for fast text revision. Our workbench features a straight-forward function to run search and replacement rules on the fly. Whenever a new replacement rule is created, it is automatically populated to the forthcoming predictions made by the system, so that the user only needs to specify them once.

4.5. Word Alignment Information

Alignment of source and target words is an important part of the translation process (Brown et al., 1993). In order to display the correspondences between both the source and target words, this feature was implemented in a way that every time the user places the mouse (yellow) or the text cursor (cyan) on a word, the alignments made by the system are highlighted.

4.6. E-Pen Interaction

E-pen interaction should be regarded as a complementary input rather than a complete replacement of the keyboard. The user can interact with the system by writing on a special area of the user interface. We decided to use MINGESTURES (Leiva et al., 2013), a highly accurate, high-performance gestures for interactive text editing.

Although in principle it would be interesting to allow the user to introduce arbitrary strings and gestures, in this approach we have decided to focus on usability. We believe that a fast response and a good accuracy are critical for user acceptance.

4.7. Logging and Replay

The workbench implements detailed logging of user activity, which enables both automatic analysis of translator behavior by aggregating statistics and enabling replay of a user session. Replay takes place in the translate view of the UI, it shows the screen at any time exactly the way the user encountered it when she interacted with the tool.

4.8. Eye-Tracking

One of the core goals of the CASMACAT project is the study of translator behavior. To better observe the activities of translators, we use eye tracking. This allows us to detect and record the exact screen position of the current focus of attention. Alongside the other logged information such as key logging, enables *translation process study*, i.e., the analysis of the behavior of the translator, opposed to just *translation product study*, i.e., the analysis of the final translation.

Eye tracking is integrated into the CASMACAT workbench using a special plugin for the browser. With this plugin, the eye tracking information is accessible to the

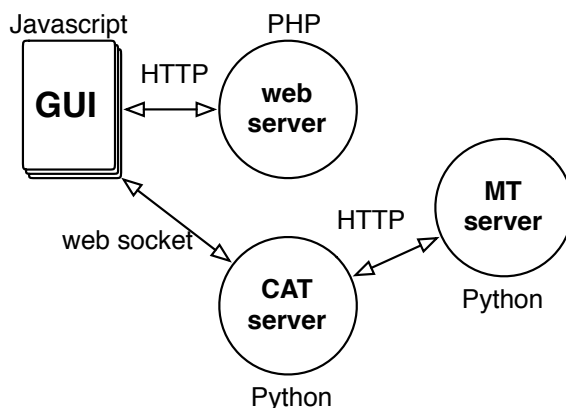


Figure 3. Modular design of the workbench: Web-based components (GUI and web server), CAT server and MT server are independent and can be swapped out

Javascript GUI and can be sent to the web server to be stored for later analysis. The eye tracking information is also visualized in the replay mode.

5. Implementation

The tool is developed as a web-based platform using HTML5 and Javascript in the Browser and PHP in the backend, supported by a CAT and MT server that run as independent process (both implemented in Python but integrating tools written in various other programming languages).

The overall design of the *CASMACAT* workbench is very modular. There are three independent components (see also Figure 3): the GUI/web server, the CAT server and the MT server. We modularize these components by clearly specified API calls, so that alternative implementations can be used as well.

5.1. Computer Aided Translation Server

The computer aided translation (CAT) server is implemented in Python with the Tornado library. It uses *socket.io* to keep a web socket connection with the Javascript GUI. Keep in mind that especially interactive translation prediction requires very quick responses from the server. Establishing an HTTP connection through an Ajax call every time the user presses a key would cause significant overhead.

A typical session with interactive translation prediction takes place as follows:

- The user moves to a new segment in the GUI.
- The GUI sends a **startSession** request to the CAT tool, passing along the input sentence.
- The GUI and CAT server establish a web socket connection.

- The CAT server requests and receives from the MT server the sentence translation and the search graph.
- The CAT server sends back the translation to the GUI and keeps the search graph in memory.
- The user starts typing (approving some of the translation or making corrections).
- At each key stroke, the GUI sends a request to the CAT server, for instance requesting a new sentence completion prediction (**setPrefix**).
- The CAT server uses the stored search graph to compute a new prediction and passed it back to the GUI (**setPrefixResult**).
- The GUI displays the new prediction to the user.
- Eventually, the user leaves the segment.
- The GUI sends a **endSession** request to the CAT tool.
- The CAT server discards all temporary data structures.
- The GUI and CAT server disconnect the web socket connection.

5.2. Machine Translation Server

For many of the CAT server's functions, information from the Machine Translation (MT) server is required. This includes not only the translation of the input sentence, but also n-best lists, search graphs, word alignments, etc.

The main call to the server is a request for a translation. The request includes the source sentence, source and target language, and optionally a key identifying the user. The server responds to requests with a JSON object, for instance:

```
{
  "data": {
    "translations": [
      {
        "sourceText": "test",
        "translatedText": "testo",
        "tokenization": {
          "src": [[0, 3]],
          "tgt": [[0, 4]]
        }
      }
    ]
  }
}
```

Note that this is based on the API of Google Translate. Our server implementation extends this API in various ways, such as the provision of aforementioned additional information, requests for tokenization and detokenization, etc.

6. Installation

Instructions are available online on the CASMACAT web site³ how to install the workbench on a consumer-grade computer running Linux.

The CASMACAT workbench uses a standard set of tools: the Apache web server, the programming language PHP, and the MySQL database. All these tools are part of a standard Linux distribution but may need to be installed on demand. The most computationally demanding process will be training a machine translation system on large amounts of data.

6.1. Web Server

The main server component is the CASMACAT web server that runs under Apache and provides the user interface over any internet browser.

Download the Source Code First find a suitable directory for the CASMACAT code. If you install it as a user, you can place it in your home directory. If you install it as administrator, you may choose something like `/opt/casmacat`

In that directory, type:

```
git clone git://git.assembla.com/matecat_source.git web-server
cd web-server
git checkout casmacat
```

You will find additional installation instructions in the file `INSTALL.txt`. It may be more up-to-date and contain more information than the instructions below.

Create a Database The files `lib/model/matecat.sql` and `lib/model/casmacat.sql` contain the configuration for the database. You may want to edit these files to change the name of the database, which by default is `matecat_sandbox`. If you do so, please change both files. You will also need to set up a user for the database. There may be a GUI in your Linux distribution to make this easy, otherwise you can call MySQL from the command line:

```
mysql -u root -p
mysql> connect mysql;
mysql> create user johndoe@localhost identified by 'secretpw';
mysql> create database matecat_sandbox;
mysql> grant usage on *.* to johndoe@localhost;
mysql> grant all privileges on matecat_sandbox.* to johndoe@localhost;
```

³<http://www.casmacat.eu/index.php?n=Workbench.Workbench>

With user account in place and (possibly edited) configuration files, you can now set up the database:

```
mysql -u johndoe -p < lib/model/matecat.sql
mysql -u johndoe -p < lib/model/casmacat.sql
```

To complete the setup of the database, you have to create a copy of the web server configuration file and edit it to point to your database installation.

Set Up the Web Server First, you need to create a user account (such as `catuser`) and add it to the `www-data` group (as root). Apache needs to be configured to access your CASMACAT web server installation. This is done with a configuration file in `/etc/apache2/sites-available`, linked to from `/etc/apache2/sites-enabled`. This configuration file follows a template provided in the source directory. With all this, you may now restart Apache with `apache2ctl restart`. If you now point your web browser to your site, you should see the CASMACAT home page.

Test the Installation To use the tool, you will have to set up a CAT server. We describe in the next section how to do this. If you want to test your current setup, you can also use a demo CAT server at the University of Edinburgh. The installation web page shows provides the configuration files and a test document that can be used for translation.

6.2. CAT Server

The computer aided translation (CAT) server communicates with the machine translation server to provide services to the CASMACAT Workbench.

Install The CAT server is available at the following Git repository:

```
cd /opt/casmacat
git clone git://github.com/hsamand/casmacat-cat-server.git cat-server
```

Configure Currently, the CAT server is set up to only serve one language pair (and system). It calls the MT server with a HTTP request.

The configuration of the URL of the machine translation server is currently hard-coded in lines 103–106 of `cat-server.py`:

```
port = 8644
if isinstance(text, unicode):
    text = text.encode('UTF-8')
url = 'http://127.0.0.1:%d/%s?%s' % (
```

Please change these lines if you machine translation server does not reside on the same machine (127.0.0.1) or responds to a different port (8644).

Run After setting the port of the machine translation server, you can run the CAT server by specifying the port it itself is listening to `./cat-server.py 9997`.

6.3. MT Server

The CASMACAT workbench may interact with any machine translation system that responds to the API according to the specifications. In the following, we describe how to set up a machine translation server using the open source Moses system. The installation requires two parts: (1) the core Moses system, from which we will use the `mosesserver` process, and (2) a Python Server Wrapper that calls `mosesserver`, alongside other pre-processing and post-processing steps, and provides additional services.

Install the Moses Server Wrapper You can download the server script (written in Python) from its Git repository:

```
cd /opt/casmacat
git clone git://github.com/christianbuck/matecat_util.git mt-server
```

The server (in `python_server/server.py`) requires `cherryypy` to run, so you may have to install that as well.

Install Moses Installing Moses may be a longer process, please refer to the Moses web site for installation instructions. You will need `bin/mosesserver` and various scripts in the `scripts` directory to handle pre- and post-processing.

Set Up a Toy Model You can download a toy French-English system from the CASMACAT web site. The system consists of a model (`toy-fr-en`) directory and a number of support scripts (in the `scripts` directory). The system comes with the shell script `TOY.fr-en` that starts up `mosesserver` and the Python Moses server wrapper.

This script starts `mosesserver` to listen to port 9998 and the Python server wrapper to listen to port 9999. While `mosesserver` carries out the core translation task, the Python server wrapper deals with additional pre- and post-processing.

Connect the MT Server to the CASMACAT Workbench To point your CAT server to your machine translation server, you have to edit in `cat-server.py` the following lines:

```
port = 9999
if isinstance (text, unicode):
    text = text.encode ('UTF-8')
url = 'http://127.0.0.1:%d/%s?%s' % (
```

Now restart your CAT server with `./cat-server.py 9997`. You have completed the installation of the CAsMACAT Workbench.

7. Outlook

This public release of the workbench occurs at the mid-point of the project and offers basic functionality of the main components of the system. For the remainder of the project, a number of extensions will be integrated, such as the visualization of translation options, a bilingual concordancer, paraphrasing on demand. We also expect that several of the capabilities will be refined and their quality improved.

In collaboration with the MATECAT project we also expect the implementation of functionality targeted at professional users, such as better user administration and document management.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement 287576 (CASMACAT). The workbench was developed in close collaboration with the MATECAT project.

Bibliography

- Alabau, Vicent, Luis A. Leiva, Daniel Ortiz-Martínez, and Francisco Casacuberta. User evaluation of interactive machine translation systems. In *Proc. EAMT*, pages 20–23, 2012.
- Barrachina, Sergio, Oliver Bender, Francisco Casacuberta, Jorge Civera, Elsa Cubel, Shahram Khadivi, Antonio Lagarda, Hermann Ney, Jesús Tomás, Enrique Vidal, and Juan-Miguel Vilar. Statistical approaches to computer-assisted translation. *Computational Linguistics*, 35 (1):3–28, 2009.
- Brown, Peter F, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.
- Federico, Marcello, Alessandro Cattelan, and Marco Trombetti. Measuring user productivity in machine translation enhanced computer assisted translation. In *Proceedings of the Tenth Conference of the Association for Machine Translation in the Americas (AMTA)*, 2012. URL <http://www.mt-archive.info/AMTA-2012-Federico.pdf>.

- González-Rubio, Jesús, Daniel Ortiz-Martínez, and Francisco Casacuberta. On the use of confidence measures within an interactive-predictive machine translation system. In *Proc. EAMT*, 2010.
- Koehn, Philipp. Enabling monolingual translators: post-editing vs. options. In *Proc. NAACL*, pages 537–545, 2010.
- Langlais, Philippe, George Foster, and Guy Lapalme. TransType: a computer-aided translation typing system. In *NAACL Workshop: EmbedMT*, pages 46–51, 2000.
- Leiva, Luis A., Vicent Alabau, and Enrique Vidal. Error-proof, high-performance, and context-aware gestures for interactive text edition. In *Proceedings of the 2013 annual conference extended abstracts on Human factors in computing systems (CHI EA)*, pages 1227–1232, 2013.
- Plitt, Mirko and Francois Masselot. A productivity test of statistical machine translation post-editing in a typical localisation context. *Prague Bulletin of Mathematical Linguistics*, 93:7–16, 2010. URL <http://ufal.mff.cuni.cz/pbml/93/art-plitt-masselot.pdf>.
- Pouliquen, Bruno, Christophe Mazenc, and Aldo Iorio. Tapta: A user-driven translation system for patent documents based on domain-aware statistical machine translation. In Forcada, Mikel L., Heidi Depraetere, and Vincent Vandeghinste, editors, *Proceedings of the 15th International Conference of the European Association for Machine Translation (EAMT)*, pages 5–12, 2011.
- Skadiņš, Raivis, Maris Puriņš, Inguna Skadiņa, and Andrejs Vasiljevs. Evaluation of SMT in localization to under-resourced inflected language. In Forcada, Mikel L., Heidi Depraetere, and Vincent Vandeghinste, editors, *Proceedings of the 15th International Conference of the European Association for Machine Translation (EAMT)*, pages 35–40, 2011.

Address for correspondence:

Philipp Koehn
pkoehn@inf.ed.ac.uk
Informatics Forum 4.19
10 Crichton Street, Edinburgh
EH8 9AB, United Kingdom



Sequence Segmentation by Enumeration: An Exploration

Steffen Eger

Goethe University
Frankfurt am Main/Germany

Abstract

We investigate *exhaustive enumeration* and subsequent *language model evaluation* (E&E approach) as an alternative to solving the sequence segmentation problem. We show that, under certain conditions (on string lengths and regarding a possibility to accurately estimate the number of segments), which are satisfied for important NLP applications, such as phonological segmentation, syllabification, and morphological segmentation, the E&E approach is feasible and promises superior results than the standard sequence labeling approach to sequence segmentation.

1. Introduction

By *sequence segmentation*, we mean the splitting of a sequence $x = x_1 \dots x_n$ consisting of n characters, each from an alphabet Σ , into non-overlapping *segments*, or *parts*, such that the concatenation of the segments, in the ‘original’ order, precisely yields x . Usually, in applications, we do not seek an arbitrary segmentation of x but the ‘most suitable’, where suitability may be defined, particularly in a supervised setting as we consider, with respect to a given distribution of data. In NLP, segmentations of sequences may occur in a variety of contexts such as *morphological segmentation*, the breaking of words into morphemes, *syllabification*, the breaking of words into syllables, *phonological segmentation*, the breaking of words into ‘phonological units’, or *word segmentation* (cf. Goldwater et al., 2009), the breaking of sentences into words. For example, the sequence $x = \text{phoenix}$ may admit suitable segmentations as in

ph-oe-n-i-x phoe-nix phoenix

for phonological segmentation, syllabification, and morphological segmentation, respectively, and where we delineate segments in an intuitive manner.

In a supervised learning context, sequence segmentation may be considered a *sequence labeling problem* where the labels indicate whether or not a split occurs at a given character position. For instance, the above segmentations of $x = \text{phoenix}$ may be encoded as

p h o e n i x	p h o e n i x	p h o e n i x
0 0 1 0 1 1 1	0 0 0 0 1 0 0	0 0 0 0 0 0 0

where a ‘1’ indicates a split.

Alternatively, we may view sequence segmentation as an ‘evaluation’ problem, in an apparently intuitive manner. Namely, given a test string x , enumerate all possible segmentations of x and evaluate, or score, each of them using a language model trained on the training data. Such an approach is potentially superior because it allows to take a ‘word’ (rather than ‘character’) perspective on the data. Moreover and most importantly, exhaustive search for evaluation is an *exact* paradigm for *arbitrary* evaluation models, whereas sequence labeling models typically make crucial, e.g., independence assumptions on scoring functions (see our discussion in Section 5.4).

The problem with the ‘evaluation viewpoint’, and the exhaustive search it naïvely relies upon, is that there are 2^{n-1} possible segmentations of a sequence x of length n , i.e., the search space is exponential in the number of characters of x , which makes the approach apparently impractical for all but very short sequences. In the current work, we challenge this claim. We present a simple model for sequence segmentation that rests on the evaluation viewpoint outlined above and on exhaustive enumeration, and that works well, as we demonstrate, under the following two conditions,

- for a given test string x , the *number* of segments of an optimal segmentation of x is known (or known to be in a ‘small’ interval) or can easily and accurately be predicted,
- for a given test string x , the length n of x is not ‘too large’ (e.g., is certainly less than 50) and/or the possible lengths of segments are not ‘too large’ (e.g., are less than 10 or so).

As we show in the next sections, when these assumptions are satisfied, exhaustive enumeration is in fact cheap and can easily be implemented. Consequently, in this situation, it is unproblematic to apply the evaluation viewpoint to sequence segmentation, which, as we show via experiments, may yield superior results for the sequence segmentation problem; we indicate error rate decreases between 5 and 42% over state-of-the-art sequence labeling approaches across different data sets. In the current work, we demonstrate, moreover, that our two criteria outlined above apparently hold for a number of ‘string related’ sequence segmentation problems in NLP such as morphological segmentation, syllabification, and phonological segmentation (they certainly do *not* apply to, e.g., word segmentation). In this respect, hence, our methodology is apparently well suited to a class of important NLP applications.

This work is structured as follows. In Section 2, we more thoroughly investigate the search space for (naïve) sequence segmentation. We do so by referring to results on *restricted integer compositions*, a field in mathematical combinatorics that has recently

gained increasing interest (Heubach and Mansour, 2004; Bender and Canfield, 2005; Shapcott, 2012; Eger, 2013). In Section 3, we illustrate our approach in more detail, before describing our data in Section 4. Then, in Section 5, we detail our experiments on sequence segmentation. Since our approach may be prone to misinterpretation, we discuss and summarize the intentions of our approach and the lessons that can be learned from it in Section 5.4. In Section 6, we discuss related work and in Section 7, we conclude.

2. Search Space for Sequence Segmentation

We first define integer compositions and then show their relationship to sequence segmentations.

Let $n, k \in \mathbb{N} = \{0, 1, 2, \dots\}$. An *integer composition of n with k parts* is a k -tuple $(\pi_1, \dots, \pi_k) \in \mathbb{N}^k$ such that $\pi_1 + \dots + \pi_k = n$. Denote by $\mathcal{C}(n, k)$ the set of all integer compositions of n with k parts. Obviously, there exists a ‘natural’ bijection between segmentations of a sequence $\mathbf{x} = x_1 \dots x_n$ of length n with k segments and integer compositions of n with k parts in which the sizes of parts correspond to the lengths of the respective segments as in

$$\begin{array}{ccccccc} \text{p h o e n i x} \\ 7 = & 2 & + & 2 & + & 1 & + & 1 & + & 1 \end{array}$$

Thus, the number of sequence segmentations of $\mathbf{x} = x_1 \dots x_n$ with k segments equals the number of integer compositions of n with k parts, $|\mathcal{C}(n, k)| = |\mathcal{S}(n, k)|$, where $\mathcal{S}(n, k)$ denotes the set of all segmentations of $x_1 \dots x_n$ with k segments. There are several well-known combinatorial results regarding the number of integer compositions of n with k parts. For example,

$$|\mathcal{C}(n, k)| = \binom{n-1}{k-1},$$

where $\binom{n}{k}$ denotes the respective binomial coefficient. Moreover, less well-known, the number of *restricted integer compositions*, that is, where each part is restricted to lie within an interval $A = \{\xi_{\min}, \xi_{\min} + 1, \dots, \xi_{\max}\}$, $\xi_{\min}, \xi_{\max} \in \mathbb{N}$, with $\xi_{\min} \leq \xi_{\max}$, is given by the *extended binomial coefficient* (Fahssi, 2012; Eger, 2013)¹

$$|\mathcal{C}_A(n, k)| = \binom{k}{n - \xi_{\min} k}_{\xi_{\max} - \xi_{\min} + 1}, \quad (1)$$

where $\binom{k}{n}_{l+1}$ arises as the coefficient of X^n of the polynomial $(1 + X + X^2 + \dots + X^l)^k$ and where we denote by $\mathcal{C}_A(n, k)$ the set of all compositions of n with k parts, each

¹Extended binomial coefficients share many interesting properties with ordinary binomial coefficients; see the discussions in the cited works.

within the interval A . As above, it obviously holds that $|\mathcal{C}_A(n, k)| = |\mathcal{S}_A(n, k)|$, where $\mathcal{S}_A(n, k)$ is the set of all segmentations of a sequence of length n with k segments where segment lengths are restricted to lie within A . Restrictions on segment lengths may be useful and justified in NLP applications; for instance, in phonological segmentation, we would hardly expect a segment to exceed, say, length 4,² and in syllabification, syllables that exceed, say, length 9 or 10, are presumably very rare across different languages.

As concerns the total number of sequence segmentations of a sequence of length n , we have

$$|\mathcal{S}(n)| = |\mathcal{C}(n)| = \sum_{k \geq 1} \binom{n-1}{k-1} = 2^{n-1},$$

where we use analogous notation as above. For restricted sequence segmentations, closed form formulas are more difficult to obtain. For $A = \{1, \dots, b\}$, $|\mathcal{S}_A(n)|$ is a *generalized Fibonacci number* satisfying the recurrence

$$|\mathcal{S}_{\{1, \dots, b\}}(n)| = \sum_{i=1}^b |\mathcal{S}_{\{1, \dots, b\}}(n-i)|.$$

Asymptotic formulas are given, e.g., by Malandro (2011) as

$$|\mathcal{S}_{\{1, \dots, b\}}(n)| \sim \frac{\phi^{n+1}}{G'(1/\phi)}, \quad (2)$$

where ϕ is the unique positive real solution to $\sum_{i=1}^b X^{-i} = 1$ and where $G(X) = \sum_{i=1}^b X^i$ and G' denotes its first derivative. For instance, there are 5 restricted segmentations of $\mathbf{x} = x_1 x_2 x_3 x_4$ where $A = \{1, 2\}$ — namely, $x_1 x_2 - x_3 x_4$; $x_1 x_2 - x_3 - x_4$; $x_1 - x_2 x_3 - x_4$; $x_1 - x_2 - x_3 x_4$; and $x_1 - x_2 - x_3 - x_4$ — while the approximation formula gives the (very close) value 4.96 for $|\mathcal{S}_{\{1, 2\}}(4)|$, since $\phi = (1 + \sqrt{5})/2$ in this case. Formula (2) also indicates that the number of segmentations of a sequence \mathbf{x} asymptotically grows *exponentially* in the length n of \mathbf{x} , even under restrictions on segment sizes, although, for any given n , there might be much fewer restricted segmentations than in the unrestricted case. For example, $|\mathcal{S}_{\{1, 2\}}(15)| = 987$, while $|\mathcal{S}(15)| = 2^{14} = 16384$.

Efficient algorithms for generating restricted integer compositions have recently been suggested in Opdyke (2010); Page (2012) and a Matlab implementation of the algorithm designed in Opdyke (2010) is available from <http://www.mathworks.com/matlabcentral/fileexchange/27110-restricted-integer-composition>.

²See Table 1 for examples.

3. Method

As we have indicated, our approach for (supervised) sequence segmentation is as follows. Given labeled data (i.e., with ‘gold standard’ segmentations), at *training time*, we simply train a language model LM on the training data set. At *test time*, we predict the segmentation of a test string x by exhaustively enumerating all possible segmentations of x and evaluating each of them via LM. The best scoring segmentation is then our prediction for x . We refer to this approach as E&E (for ‘enumerate and evaluate’). As mentioned, since enumerating (really) *all* possible segmentations of x is generally impracticable (even for restricted segmentations), we crucially rely on a ‘number of parts’ prediction model PM; predicting the number of parts of the correct segmentation of x is a simpler problem than actually providing the correct segmentation. We outline a possible strategy for specifying PM below.

We consider both a *word level*, LM-W, and a *character level*, LM-C, language model for our E&E approach. The character level model views (training) strings as a sequence of ‘characters’ as in *ph-oe-n-i-x* (including the split information) while the word level model views the same strings as a sequence of ‘words’ as in *ph oe n i x* (which also includes the split information). Intuitively, we would expect both models to perform differently in different situations. For example, in syllabification, segmentations crucially depend on character information (e.g., whether or not the current character is a vowel or a consonant) while in word segmentation or morphological segmentation, a word level view may be a ‘superior’ perspective.

4. Data and Its Statistical Properties

We use CELEX (Baayen et al., 1996) as our lexical database. CELEX provides information on orthographical (syllabification) and morphological segmentation for German, English, and Dutch. Moreover, it provides phonological transcriptions for the three languages. To generate phonological segmentations from these, we first align words with their phonological representations via a monotone many-to-many aligner (cf. Eger, 2012) and then retrieve the phonologically segmented words. For the phonology data, we use random subsets of data from the Pascal challenge (Van den Bosch et al., 2006), which, in the case of German and Dutch, is directly based on CELEX but already provides a filtering; here, we also include data on French from the Pascal challenge, which is based on the Brulex database (Content et al., 1990). In the case of orthographical and morphological segmentation, we remove all duplicates and multi-word entries from CELEX and focus on random subsets of given sizes, as indicated in Table 2. In Table 1, we give examples of gold standard segmented data, across the different languages and segmentation domains.

Table 2 summarizes statistical properties of our data sets. The first three columns refer to the minimum, maximum, and average number of parts of segmentations in the various gold standard alignments. The next three columns refer to the minimum,

G-P	b-e-r-ei-t, sch-uh, sch-n-ee-m-a-tsch, s-ä-tt-i-g-u-ng
E-P	ear-th-en, th-r-ough, o-ff-sh-oo-t, a-gg-r-e-ss-i-ve
D-P	sj-o-tt-en, w-ij-n-h-ui-s, i-mm-uu-n, p-r-ui-s-i-sch
F-P	s-aint, e-rr-an-ce, r-a-b-a-tt-eu-r, b-u-r-eau-c-r-a-te
G-S	a-so-zi-a-le-re, e-be-ne, schnee-sturms, schnupft
E-S	bo-liv-i-a, id-i-ot, ring-side, scrunched
D-S	i-ni-ti-a-le, maan-zie-ke-re, kerst-staaf, traagst
G-M	er-barm-ung-s-los-ig-keit, titel-schrift, kinkerlitzchen
E-M	un-profess-ion-al-ly, im-patient-ly, un-do, quincenary

Table 1. Examples of gold standard segmentations from different data sets. In the first column, G,E,F, and D stand for German, English, French, and Dutch, respectively. P, S, and M stand for phonology, syllabification, and morphology data, respectively.

maximum, and average sizes of the parts and the subsequent three columns to the minimum, maximum, and average string lengths. The last three columns give numbers relating to the size of the search space for full enumeration, which we determine via relationship (1). As concerns the size of the search space, i.e., the number of possible segmentations of strings x under these parameter values, we find that the number $S_A([\bar{n}], [\bar{k}])$, which gives the number of segmentations of the *average* string with an *average* number of parts, is usually quite small, ranging from 7 to 120 across the different data sets. Also, the 95 percent quantiles show that 95 percent of all strings, across the different datasets, admit at most a few hundred or a few thousand segmentations. The expected values are also moderate in size but the large standard deviations indicate that the distributions of the number of segmentations per string is very skewed, where a few strings allow very many segmentations. For example, the German noun *wahrscheinlichkeitsrechnung*, with length $n = 27$, admits 2,653,292 segmentations with $k^* = 18$ parts, each between $\xi_{\min} = 1$ and $\xi_{\max} = 4$.

5. Experiments

For our experiments, we use as language model LM standard Ngram models, with modified Kneser-Ney smoothing, as implemented in the kylm language modeling toolkit.³ We emphasize that we choose (discrete) Ngram models as language models merely for the sake of convenience and because Ngram models have a very strong tradition in NLP; other language models such as log-linear language models (Berger et al., 1996) or neural network language models (Bengio et al., 2001) might have been equally good (or better) alternatives. To contrast our methodology with the sequence labeling approach to sequence segmentation, we use conditional random fields (CRFs)

³Available at <http://www.phontron.com/kylm/>.

	k_{\min}	k_{\max}	\bar{k}	ξ_{\min}	ξ_{\max}	$\bar{\xi}$	n_{\min}	n_{\max}	\bar{n}
G-P-25K	1	27	8.66 ± 2.7	1	4	1.15 ± 0.4	1	31	9.97 ± 3.1
E-P-25K	1	20	7.18 ± 2.3	1	4	1.17 ± 0.4	1	22	8.37 ± 2.5
D-P-25K	1	25	9.09 ± 3.1	1	4	1.16 ± 0.4	1	29	10.53 ± 3.5
F-P-25K	1	18	6.69 ± 2.3	1	4	1.27 ± 0.5	1	20	8.50 ± 2.6
G-S-55K	1	10	3.62 ± 1.2	1	10	3.08 ± 1.1	1	31	11.15 ± 3.2
E-S-15K	1	7	2.43 ± 1.1	1	9	3.20 ± 1.3	1	19	7.80 ± 2.5
D-S-55K	1	11	3.51 ± 1.3	1	9	3.07 ± 1.0	1	30	10.78 ± 3.3
G-M-36K	1	9	2.40 ± 0.9	1	21	4.17 ± 2.1	1	31	10.01 ± 3.2
E-M-22K	1	5	1.68 ± 0.7	1	16	4.60 ± 2.1	1	21	7.73 ± 2.6

	$ S_A([\bar{n}], [\bar{k}]) $	$ Q_{[S_A(n,k)]}^{0.95} $	$E[S_A(n, k)]$
G-P-25K	9	364	$465.52 \pm 28,058.0$
E-P-25K	7	105	27.55 ± 98.7
D-P-25K	45	364	$238.06 \pm 4,853.2$
F-P-25K	28	286	78.08 ± 504.8
G-S-55K	120	2,710	$1,547.04 \pm 56,553.2$
E-S-15K	7	210	59.99 ± 318.2
D-S-55K	120	2,430	$2,848.16 \pm 75,541.1$
G-M-36K	9	364	$365.17 \pm 31,063.1$
E-M-22K	7	45	10.43 ± 30.6

Table 2. Data sets (rows) and statistical properties. The set A is $\{\xi_{\min}, \xi_{\min} + 1, \dots, \xi_{\max}\}$. Description in text.

(Lafferty et al., 2001) as a sequence labeling model SL, as implemented in the CRF++ toolkit.⁴ Again, alternatives such as structured SVMs (Tsochantaridis et al., 2004) might have been equally well (or better) suited but we choose CRFs because of their reputation as yielding state-of-the art results on structured prediction problems in NLP. For all subsequent experiments, we use linear-chain conditional random fields with window size w (we include as features all character Ngrams that fit inside a window of $\pm w$ around the current character).

In our sequence labeling approach, we additionally consider another encoding scheme as the one indicated in Section 1. Namely, we also experiment on encoding the length of the segment directly in the labeling. For example, for the syllabic segmentation of *phoenix* as given in Section 1, this labeling would read as $0_1 0_2 0_3 0_4 10_1 0_2$ to represent the segmentation *phoen-ix*. Bartlett et al. (2008) have claimed that this *numbered encoding scheme* leads to better performance for the syllabification problem

⁴Available at <http://crfpp.googlecode.com/svn/trunk/doc/index.html>

because it biases the model to favor shorter segments. We refer to this labeling scheme as SL-NUM and to the (unnumbered) labeling scheme outlined in Section 1 as, simply, SL.

Generally, for all subsequent experiments, when indicating a dataset of size M , we perform ten-fold cross-validation to assess performance results, that is, our training data has size $0.9M$ for each of the ten folds. Throughout, as a performance measure, we use *word error rate*, the fraction of wrongly segmented sequences.

5.1. Phonological Segmentation

For phonological segmentation, we generate random samples of size $M = 25,000$ for German, English, Dutch, and French in the manner indicated in Section 4. We first assess, in Table 3, how well our SL modeling performs as a part prediction model PM. We see that k^* , the true number of parts of a given sequence, on average coincides with \hat{k} , the predicted number of parts, in about 97% of the cases for German, Dutch, and French, and in about 91% of the cases for English. Thus, if we used our LM models with \hat{k} , we would have error rates of at least 3% for German, Dutch, and French, and at least 9% for English. Higher upper bounds on performance can be reached by instead considering the intervals $B_1(\hat{k}) = \{\hat{k} - 1, \hat{k}, \hat{k} + 1\}$ wherein to search for k^* . In fact, as shown in the table, the probability that k^* is in $B_1(\hat{k})$ is considerably above 99% for all four datasets. These findings encourage us to use our *sequence labeling models SL as prediction models PM*.

	$P_{SL}[k^* = \hat{k}]$	$P_{SL}\left[k^* \in \{\hat{k} - 1, \hat{k}, \hat{k} + 1\}\right]$
German-25K	97.5 ± 0.25	99.8 ± 0.13
English-25K	90.9 ± 0.44	99.3 ± 0.27
Dutch-25K	96.5 ± 0.29	99.9 ± 0.08
French-25K	97.1 ± 0.26	99.9 ± 0.08

Table 3. Probability that k^* is identical to \hat{k} as predicted by SL model or is in $B_1(\hat{k})$ in %. Phonology data.

Next, in Figure 1, we plot error rates in terms of N (for the LM Ngram models; we use \hat{k} from the SL models). We see that for the LM-C models, performance levels off at about $N = 10$ or $N = 11$, while for the LM-W models, performance levels off already at $N = 6$ or $N = 7$. This is understandable as the word level models operate on entities of a larger size, namely, segments. We also usually see a convergence of both error rates as N gets larger. We omit similar graphs for window sizes w in the SL models,

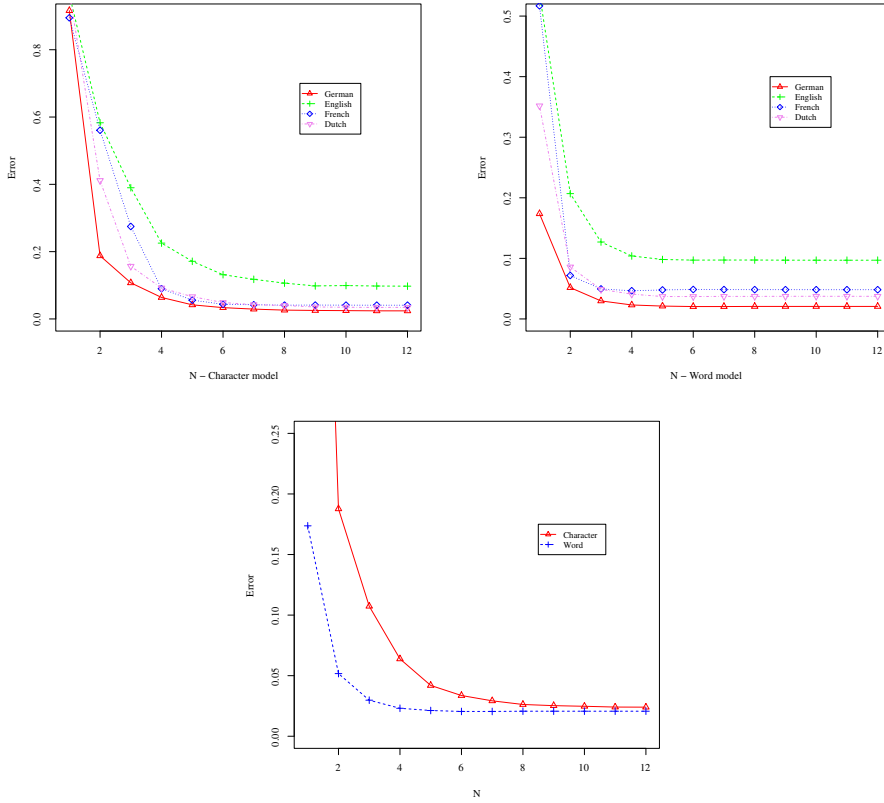


Figure 1. Performance of LM-C (top left) and LM-W (top right) as a function of N in the Ngrams. Bottom: Character and word model in a single plot, exemplarily shown for German. Phonology data.

but remark that a leveling off of performance occurs usually at $w = 4$ (note that this means that a context of $2w + 1 = 9$ characters is considered) or $w = 5$.

Now, based on these insights, we fix N at 10 for the LM-W models and at 11 for the LM-C models and use $w = 5$ for the SL models. We report results in Table 4. Throughout, we see that on our four datasets, the models SL-NUM and SL have no statistically significant different error rates, that is, on this data and for our CRF learning models, we cannot confirm that using the numbered coding scheme implies better performance results. Moreover, the two LM models have no statistically significant better performance than the SL models, too, when using as prediction for the number of parts the variables \hat{k} from the SL models. In contrast, when enumerating all

	German-25K	French-25K	Dutch-25K	English-25K
SL- $\text{NUM}_{w=5}$	2.65 ± 0.27	3.64 ± 0.31	3.91 ± 0.32	10.10 ± 0.45
SL- $w=5$	2.68 ± 0.26	3.56 ± 0.27	3.86 ± 0.29	10.07 ± 0.45
LM-C $_{N=11}^{k=\hat{k}}$	2.73 ± 0.28	3.55 ± 0.27	3.87 ± 0.30	10.05 ± 0.49
LM-W $_{N=10}^{k=\hat{k}}$	2.74 ± 0.31	3.56 ± 0.25	3.88 ± 0.29	10.06 ± 0.40
LM-C $_{N=11}^{k \in \{\hat{k}-1, \hat{k}, \hat{k}+1\}}$	2.41 ± 0.32	4.09 ± 0.26	3.42 ± 0.32	9.78 ± 0.35
LM-C $_{N=11}^{k \in \{\hat{k}-1, \hat{k}, \hat{k}+1\}, \beta=1.1}$	2.29 ± 0.27	3.39 ± 0.24	3.34 ± 0.31	9.15 ± 0.41
LM-W $_{N=10}^{k \in \{\hat{k}-1, \hat{k}, \hat{k}+1\}}$	2.06 ± 0.37	4.83 ± 0.33	3.74 ± 0.30	9.70 ± 0.48
LM-W $_{N=10}^{k \in \{\hat{k}-1, \hat{k}, \hat{k}+1\}, \beta=1.1}$	2.09 ± 0.26	3.65 ± 0.31	3.44 ± 0.34	9.03 ± 0.49
LM-C $_{N=11}^{k=k^*}$	0.63 ± 0.25	1.22 ± 0.21	1.21 ± 0.09	3.01 ± 0.39
LM-W $_{N=10}^{k=k^*}$	0.60 ± 0.21	1.22 ± 0.19	1.23 ± 0.16	3.04 ± 0.35

Table 4. Error rates in % across different data sets and model specifications. Sub- and superscripts denote various parameterizations. Phonology data.

segmentations with number of parts k in $B_1(\hat{k})$ and selecting the highest scoring as predicted segmentation, performance results are, except for the French Brulex data, significantly better. For example, for the German, English and Dutch data, we find, for LM-C, error rate improvements of about 10%, 3%, and 11%, with regard to the SL models. Still larger improvements can be achieved by putting a *prior* on k . Note that, since the SL models are quite accurate PM models, it is more likely that \hat{k} is correct than either $\hat{k} - 1$ or $\hat{k} + 1$. We experiment with a very simple heuristic that discounts the language model likelihood of segmentations with $\hat{k} \pm 1$ parts by a factor β . While selecting β by optimizing on a development set might lead to best results, we simply let $\beta = 1.1$ for all our data sets. This implies error rate improvements of about, for our best LM-C or LM-W models, 22%, 10%, 13%, and 5% for German, English, Dutch, and French, respectively, with respect to the SL models, where all improvements are statistically significant (paired two-tailed t-test, 5% significance level). Finally, as a kind of ‘oracle model’, we give performance results of our LM models under the assumption that the true number of parts k^* were known, for each given string to segment. We see, in this case, very large error rate improvements, of about 77%, 68%, 69%, and 65% for German, English, Dutch, and French, respectively, with respect to the SL models.

To say a word on the difference between the LM-C and LM-W models, we find that, a bit surprisingly, both models apparently perform, more or less, equally well (we would have expected the word level models to outdo the character level models, at

least on the phonological segmentation task). In our case, the word level models perform better for German and slightly better for English, while this ordering is reversed for French and Dutch.

As concerns *running times*, on a 3.1 GHz processor, it takes around 18 min, over all 10 folds, for the CRFs to train, both for English (smallest string lengths, on average) and Dutch (largest string lengths, on average). Testing (decoding) takes about 2.39s for English and 3.69s for Dutch. In contrast, training the LM models takes around 42s for English and around 52s for Dutch. Generating all segmentations and evaluating them takes around 22s + 2min for English and 14min + 25min for Dutch when choosing $B_1(\hat{k})$ as search space. Thus, all in all, running times are quite moderate; also note that our segmentation and evaluation module are in Matlab (resp. Python) and Java, whereas the CRF is in C++. We also find that we search about 0.77% of the full search space (2^{n-1} segmentations per string of length n) and that if we explored the full search space, running times would be inflated by a factor of about 130 (hence, segmenting and evaluating 25,000 strings would take about 3 1/2 days for the Dutch data), with no (or almost no) increase in accuracy because $B_1(\hat{k})$ contains all (or almost all) correct segmentations (in fact, switching to, e.g., $B_2(\hat{k})$ implies no statistically distinguishable performance results, as we find).

We are not aware of any other study that would evaluate phonological sequence segmentation (but see also the related work section) and thus cannot compare our results here with those of others.

5.2. Syllabification

For syllabification, we use data set sizes as reported in Bartlett et al. (2008). In Table 5, we see that our SL model performs better here in predicting the correct number of parts of segmentations than in the phonological segmentation task, where the probability that the true k^* is in $B_1(\hat{k})$ is very close to 100% across the three languages.

	$P_{SL}[k^* = \hat{k}]$	$P_{SL}\left[k^* \in \left\{\hat{k} - 1, \hat{k}, \hat{k} + 1\right\}\right]$
German-55K	99.6 ± 0.09	100.0 ± 0.00
English-15K	96.7 ± 0.52	99.9 ± 0.03
Dutch-55K	99.4 ± 0.07	99.9 ± 0.01

Table 5. Probability that k^* is identical to \hat{k} as predicted by SL model or is in $B_1(\hat{k})$ in %. Syllabification data.

While we omit an investigation of varying N in the Ngram models because of similarity of graphics with those previously shown, we mention that increasing N above

2 or 3 has no impact in the LM-W models since the average number of parts is much smaller here than in the phonological segmentation case (see Table 2); the same holds true for the morphological segmentation task below.

Thus, we fix N at 3 in the LM-W models and at 11, as before, in the LM-C models, giving results in Table 6. Again, we see performance increases of about, for German, English, and Dutch, respectively, 30%, 7%, and 42% for the best performing LM models over the SL models. Knowing the true k^* would, as before, yield still considerably better results. We report on an evaluation of the word level model only in the situation of a closed language model where the vocabulary stems from the training data (this excludes on the order of 5–10% of all test strings because some of their syllable parts never occurred in the training data, no matter the possible segmentation); in fact, the open vocabulary situation is uninformative since the LM-W model has huge error rates here, as our language model reserves so much probability mass for unseen words that segmentations with segments that do not occur in the training data are favored over those whose segment parts do occur there.⁵ As we have expected, under the same conditions, the character level model still performs significantly better than the word level model in the case of syllabification. We omit an investigation of the numbered coding scheme, except for the English data, because of the huge increase in training time and since we find that this model actually never performs better than its unnumbered alternative.

Our results compare favorably with those reported by Bartlett et al. (2008), who claim to improve on competitors by a wide margin. Using an SL approach with a structured SVM as a labeling model, they obtain error rates of 1.19%, 10.55% (they give a result of 15.03% for SbA (Marchand et al., 2007)), and 1.80% for German, English, and Dutch, while we obtain 1.07%, 11.24%, and 1.49% here, with our best models. Thus, except for the English data, our results appear better, using the same training set sizes.⁶ Concerning other results, Bartlett et al. (2008) cite error rates of Bouma (2003), using finite state techniques, of 3.50% for Dutch on 50K training instances, as we use, and 1.80% on 250K. For German, Demberg (2006)’s HMM approach achieves 2.13% on the *whole* of CELEX, which is double of our error rate. To our knowledge, our results are the best reported on the syllabification task for German and Dutch on the CELEX data and for our training set size of 50K.

5.3. Morphological Segmentation

Performance results for morphological segmentation are listed in Tables 7 and 8 (the Dutch data was unfortunately not available to us here). Again, our best perform-

⁵The same does *not* hold true for phonological segmentation, where parts are shorter and strings have more segments such that more reliable statistics can be computed.

⁶ The better performance of Bartlett et al. (2008) on English may be due to the advantage of SVMs over standard Ngrams (and CRFs) at the small training set size for English; see He and Wang (2012) and our discussion below.

	German-55K	English-15K	Dutch-55K
SL-NUM _{w=5}	na	12.88 ± 0.70	na
SL _{w=5}	1.54 ± 0.21	12.14 ± 0.59	2.57 ± 0.16
LM-C _{N=11} ^{k=\hat{k}}	1.20 ± 0.17	11.73 ± 0.69	1.63 ± 0.14
LM-W _{N=3} ^{k=\hat{k}}	na	na	na
LM-C _{N=11} ^{k$\in\{\hat{k}-1, \hat{k}, \hat{k}+1\}$}	1.41 ± 0.17	12.21 ± 0.72	1.77 ± 0.12
LM-C _{N=11} ^{k$\in\{\hat{k}-1, \hat{k}, \hat{k}+1\}, \beta=1.1$}	1.07 ± 0.15	11.24 ± 0.62	1.49 ± 0.06
LM-C _{N=11} ^{k=k*}	0.82 ± 0.11	9.40 ± 0.58	1.14 ± 0.08
LM-C _{N=11} ^{vocab, k=\hat{k}}	1.49 ± 0.13	14.95 ± 0.92	1.71 ± 0.16
LM-W _{N=3} ^{vocab, k=\hat{k}}	3.53 ± 0.22	18.82 ± 1.59	3.49 ± 0.23

Table 6. Error rates in % across different data sets and model specifications. Sub- and superscripts denote various parametrizations. Syllabification data.

	German-36K	English-22K
SL-NUM _{w=5}	na	13.45 ± 0.31
SL _{w=5}	16.34 ± 0.43	11.68 ± 0.50
LM-C _{N=11} ^{k=\hat{k}}	15.15 ± 0.60	11.18 ± 0.47
LM-W _{N=3} ^{k=\hat{k}}	na	na
LM-C _{N=11} ^{k$\in\{\hat{k}-1, \hat{k}, \hat{k}+1\}$}	11.08 ± 0.49	9.50 ± 0.72
LM-C _{N=11} ^{k$\in\{\hat{k}-1, \hat{k}, \hat{k}+1\}, \beta=1.1$}	11.86 ± 0.60	9.31 ± 0.69
LM-C _{N=11} ^{k=k*}	2.31 ± 0.34	0.98 ± 0.13
LM-C _{N=11} ^{vocab, k=\hat{k}}	6.85 ± 0.68	3.60 ± 0.38
LM-W _{N=3} ^{vocab, k=\hat{k}}	6.88 ± 0.70	3.62 ± 0.40

Table 7. Error rates in % across different data sets and model specifications. Sub- and superscripts denote various parametrizations. Morphology data.

ing LM models are about 32% and 20% better, for German and English, respectively, than the SL approach. Concerning error rates, we omit a comparison with other work

because most approaches in morphological segmentation are unsupervised, and we in fact are not aware of supervised performance results for the data we consider.

	$P_{SL}[k^* = \hat{k}]$	$P_{SL}\left[k^* \in \{\hat{k} - 1, \hat{k}, \hat{k} + 1\}\right]$
German-36K	85.6 ± 0.44	98.9 ± 0.17
English-22K	89.1 ± 0.49	99.7 ± 0.05

Table 8. Probability that k^* is identical to \hat{k} as predicted by SL model or is in $B_1(\hat{k})$ in %. Morphology data.

5.4. Discussion

To say a word on exhaustive enumeration as a solution technique to optimization problems, beginner’s courses to combinatorial optimization usually emphasize that exhaustive search is the *simplest* and *most straightforward* approach to any optimization problem that admits only finitely many possible solution candidates; and that it is, if feasible at all (i.e., from a complexity perspective), also guaranteed to lead to *optimal solutions* (Nievergelt, 2000). Hence, if the segmentation problem in NLP was framed as the problem of finding the segmentation s_n of sequence $x = x_1 \dots x_n$ that solves

$$\arg \max_{s_n \in \mathcal{S}(n)} f_\theta(s_n),$$

i.e., for model f_θ and model parameter vector θ given (if one wants so, this is the *decoding problem* for sequence segmentation), then our approach to sequence segmentation would surely be optimal, provided that our search space restrictions are not critical. The problem in natural language processing (NLP) is, of course, that we neither know the most appropriate (or ‘true’) model f_θ for our task nor, in statistical NLP, do we know the true parameter vector θ . The scope of this work is neither model selection nor feature engineering (determination of a good model f_θ), however, nor is it the estimation of the parameter vector θ . What we intend to show, instead, is that, for our problem tasks, efficient enumeration is generally feasible such that, for f_θ given, our approach is optimal. Thus, to summarize, if a technique performed better than the approach sketched in this work, it must be due to a superior model f_θ (e.g., than our standard Ngrams),⁷ and not due to *search*, as we focus on. Here, we content ourselves, however, with the fact that standard Ngrams in conjunction with (almost) exact search can, as shown, outperform state-of-the-art approaches to sequence segmentation (this includes, at least on two out of the three data sets on the syllabification task, structured SVMs, which appear to be the *primus inter pares* among current

⁷Or due to ‘better’ estimation of θ .

sequence labeling methods; see the discussion below), rendering the investigation of better models f_θ momentarily superfluous.

To contrast our approach with other methods, many sequence labeling algorithms, for example, rely on crucial *restrictions with regard to allowable scoring functions* f_θ , as mentioned. For example, most graphical models assume Markov-type independence assumptions for the label sequences. In contrast, with our approach, f_θ may be arbitrary, and arbitrarily complex. To make this feasible, we instead *restrict search space*, as outlined. Moreover, as Tables 3, 5, and 8 demonstrate, the search space we prune away has very little probability of actually containing the correct segmentations (we could easily lower this probability to zero by, e.g., considering the search spaces $B_2(\hat{k})$) such that our restrictions may not affect accuracy at all, while pruning model complexity may be more expected to yield sub-optimal performance. Our approach may also be seen in the context of *coarse-to-fine decoding* procedures: first, we use a sub-optimal model f_θ^1 to restrict search space, and then use any arbitrary, ‘superior’ models f_θ^2 in conjunction with full enumeration on the restricted search space to improve on f_θ^1 ; we have shown how and that such a procedure can be made effective within the field of sequence segmentation for selected NLP applications.

We also note that for specific f_θ , e.g., when f_θ is *decomposable* (Terzi, 2006), full enumeration may not be necessary because efficient dynamic programming (DP) solutions apply. For example, for word level Ngrams, a simple DP solution whose running time is quadratic in n , sequence length, can be given when $N = 1$. In contrast, our approach works for *any* f_θ , not only for decomposable models.

6. Related Work

Phonological segmentation may be a crucial step in, e.g., grapheme-to-phoneme conversion (G2P) models based on many-to-many alignment approaches (Jiampojamarn et al., 2007, 2008; Bisani and Ney, 2008; Eger, 2012), where, for decoding, grapheme strings need to be segmented. Jiampojamarn et al. (2007) employ instance-based learning for this ‘letter chunking task’, without, however, evaluating their model’s performance (they solely evaluate G2P performance); the same holds true for the three other papers cited. Of course, sequence segmentation similar to phonological segmentation may play a key role in string transduction problems, including lemmatization, stemming, etc., in general (Dreyer et al., 2008). As concerns syllabification, besides ‘rule-based approaches’ (see the discussion and references in Marchand et al., 2007), in the statistical context, we are aware of Bartlett et al. (2008)’s sequence labeling approach and a lazy learning segmentation-by-analogy framework due to Marchand et al. (2007); older approaches include neural network backpropagation learning (Daelemans and van den Bosch, 1992) or finite-state techniques (Bouma, 2003). Intriguingly, syllabification may prove beneficial for solving the G2P task, as Bartlett et al. (2008) demonstrate; its most obvious application is, of course, to provide candidates for hyphenation. There is a huge literature on morphological segmentation,

e.g., Creutz and Lagus (2007); Poon et al. (2009), but most approaches are unsupervised here. As concerns applications of morphological segmentation, besides serving for quantitative analyses such as morpheme counts in texts, it may serve as a preprocessing step for phonological segmentation and/or syllabification.

The literature on CRFs, as we have used as a SL model, is vastly expanding, too; among the most interesting developments in our context are probably semi-Markov CRFs (Sarawagi and Cohen, 2004), which explicitly segment the input sequence. An analysis within our context would be scope for future research. Stoyanov and Eisner (2012) discuss approximate inference and decoding for higher-treewidth graphical models underlying CRFs. A recent comparison of state-of-the-art sequence labeling approaches is given in He and Wang (2012) where it is shown that structured SVMs outperform competitors on tagging and OCR; performance differences decrease, however, in data set size.

7. Concluding Remarks

Our contribution to the *mathematics* of linguistics is to relate the sequence segmentation problem to restricted integer compositions, which have attracted increasing interest in mathematical combinatorics recently — not the least because of their relationship to *extended binomial coefficients*. Our contribution to *computational* linguistics is to show that exhaustive enumeration of sequence segmentations is, for an array of interesting segmentation problems in NLP, cheap, given adequate restriction of search space, such that exact search for the optimal segmentations can easily be conducted, for arbitrary evaluation models f_θ . We also show that for the simple choice of f_θ as standard Ngram models, performance results on par or better than current state-of-the-art sequence labeling approaches can be achieved.

In future work, different language models f_θ , possibly including *global features*, are worthwhile investigating, among other things, as well as interpolating of character and word level language models.

Bibliography

- Baayen, R. Harald, Richard Piepenbrock, and Leon Gulikers. The CELEX2 lexical database, 1996.
- Bartlett, Susan, Grzegorz Kondrak, and Colin Cherry. Automatic syllabification with structured SVMs for letter-to-phoneme conversion. In *Proceedings of ACL-08: HLT*, pages 568–576. Association for Computational Linguistics, June 2008.
- Bender, Edward A. and E. Rodney Canfield. Locally restricted compositions, I. Restricted adjacent differences. *The Electronic Journal of Combinatorics*, 12, 2005.
- Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. In *NIPS 13*, pages 933–938, 2001.

- Berger, Adam L., Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, Mar. 1996. ISSN 0891-2017.
- Bisani, Maximilian and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Commun.*, 50(5):434–451, May 2008. ISSN 0167-6393. doi: 10.1016/j.specom.2008.01.002.
- Bouma, Gosse. Finite state methods for hyphenation. *Nat. Lang. Eng.*, 9(1):5–20, Mar. 2003. ISSN 1351-3249. doi: 10.1017/S1351324903003073.
- Content, Alain, Philippe Mousty, and Monique Radeau. Brulex. Une base de données lexicales informatisée pour le français écrit et parlé. *L'année psychologique*, 90(4):551–566, 1990. ISSN 0003-5033. doi: 10.3406/psy.1990.29428.
- Creutz, Mathias and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Trans. Speech Lang. Process.*, 4(1):3:1–3:34, Feb. 2007. ISSN 1550-4875. doi: 10.1145/1187415.1187418.
- Daelemans, Walter and Antal van den Bosch. Generalization performance of backpropagation learning on a syllabification task. In *Proceedings of the 3rd Twente Workshop on Language Technology*, pages 27–38, 1992.
- Demberg, Vera. Letter-to-phoneme conversion for a german text-to-speech system, 2006.
- Dreyer, Markus, Jason R. Smith, and Jason Eisner. Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, EMNLP '08, pages 1080–1089, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1613715.1613856>.
- Eger, Steffen. S-restricted monotone alignments: Algorithm, search space, and applications. In *Proceedings of Coling*, 2012.
- Eger, Steffen. Restricted weighted integer compositions and extended binomial coefficients. *Journal of Integer Sequences*, 2013.
- Fahssi, Nour-Eddine. A systematic study of polynomial triangles. *The Electronic Journal of Combinatorics*, 2012.
- Goldwater, Sharon, Thomas L. Griffiths, and Mark Johnson. A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54, July 2009. ISSN 00100277. doi: 10.1016/j.cognition.2009.03.008.
- He, Zhengyan and Houfeng Wang. A comparison and improvement of online learning algorithms for sequence labeling. In *Proceedings of Coling*, 2012.
- Heubach, Silvia and Toufik Mansour. Compositions of n with parts in a set. *Congressus Numerantium*, 164:127–143, 2004.
- Jiampojamarn, Sittichai, Grzegorz Kondrak, and Tarek Sherif. Applying many-to-many alignments and Hidden Markov Models to letter-to-phoneme conversion. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2007)*, pages 372–379, Rochester, New York, Apr. 2007. Association for Computational Linguistics.

- Jiampojarn, Sittichai, Colin Cherry, and Grzegorz Kondrak. Joint processing and discriminative training for letter-to-phoneme conversion. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, pages 905–913, June 2008.
- Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.
- Malandro, Martin E. Integer compositions with part sizes not exceeding k , 2011. Preprint available at <http://arxiv.org/pdf/1108.0337.pdf>.
- Marchand, Yannick, Connie Adsett, and Robert Damper. Evaluation of automatic syllabification algorithms for English. In *Proceedings of the 6th international speech communication association (ISCA)*, 2007.
- Nievergelt, Jürg. Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power. In *Proc. Conf. on Current Trends in Theory and Practice of Informatics*, pages 18–35, 2000.
- Opdyke, John Douglas. A unified approach to algorithms generating unrestricted and restricted integer compositions and integer partitions. *Journal of Mathematical Modelling and Algorithms*, 9(1):53–97, 2010.
- Page, Daniel R. Generalized algorithm for restricted weak composition generation. *Journal of Mathematical Modelling and Algorithms (JMMA)*, pages 1–28, 2012. ISSN 1570-1166. doi: 10.1007/s10852-012-9194-4. Published online July 20.
- Poon, Hoifung, Colin Cherry, and Kristina Toutanova. Unsupervised morphological segmentation with log-linear models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-2009*, pages 209–217, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-41-1.
- Sarawagi, Sunita and William W. Cohen. Semi-Markov conditional random fields for information extraction. In *Proceedings of NIPS*, 2004.
- Shapcott, Caroline. C -color compositions and palindromes. *Fibonacci Quarterly*, 50:297–303, 2012.
- Stoyanov, Veselin and Jason Eisner. Minimum-risk training of approximate CRF-based NLP systems. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT '12*, pages 120–130, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. ISBN 978-1-937284-20-6.
- Terzi, Evimaria. *Problems and Algorithms for Sequence Segmentation*. PhD thesis, University of Helsinki, 2006.
- Tsochantaridis, Ioannis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st international conference on Machine Learning (ICML)*, pages 823–830, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015341.

Van den Bosch, Antal, Stanley Chen, Walter Daelemans, Bob Damper, Kjell Gustafson, Yannick Marchand, and Francois Yvon. Pascal letter-to-phoneme conversion challenge, 2006. URL <http://www.pascalnetwork.org/Challenges/PRONALSYL>.

Address for correspondence:

Steffen Eger

eger.steffen@gmail.com

Goethe University

Grüneburgplatz 1

60323 Frankfurt am Main, Germany

**INSTRUCTIONS FOR AUTHORS**

Manuscripts are welcome provided that they have not yet been published elsewhere and that they bring some interesting and new insights contributing to the broad field of computational linguistics in any of its aspects, or of linguistic theory. The submitted articles may be:

- long articles with completed, wide-impact research results both theoretical and practical, and/or new formalisms for linguistic analysis and their implementation and application on linguistic data sets, or
- short or long articles that are abstracts or extracts of Master's and PhD thesis, with the most interesting and/or promising results described. Also
- short or long articles looking forward that base their views on proper and deep analysis of the current situation in various subjects within the field are invited, as well as
- short articles about current advanced research of both theoretical and applied nature, with very specific (and perhaps narrow, but well-defined) target goal in all areas of language and speech processing, to give the opportunity to junior researchers to publish as soon as possible;
- short articles that contain contraversing, polemic or otherwise unusual views, supported by some experimental evidence but not necessarily evaluated in the usual sense are also welcome.

The recommended length of long article is 12–30 pages and of short paper is 6–15 pages.

The copyright of papers accepted for publication remains with the author. The editors reserve the right to make editorial revisions but these revisions and changes have to be approved by the author(s). Book reviews and short book notices are also appreciated.

The manuscripts are reviewed by 2 independent reviewers, at least one of them being a member of the international Editorial Board.

Authors receive two copies of the relevant issue of the PBML together with the original pdf files.

The guidelines for the technical shape of the contributions are found on the web site <http://ufal.mff.cuni.cz/pbml.html>. If there are any technical problems, please contact the editorial staff at pbml@ufal.mff.cuni.cz.