Thot Toolkit for Statistical Machine Translation

User Manual

Daniel Ortiz Martínez
daniel.o@webinterpret.com
Webinterpret

December 2017

CONTENTS

CHAPTER 1
INTRODUCTION

Thot is an open source toolkit for statistical machine translation. Originally, Thot incorporated tools to train phrase-based models. The new version of Thot now includes a state-of-the-art phrase-based translation decoder as well as tools to estimate all of the models involved in the translation process. In addition to this, Thot is also able to incrementally update its models in real time after presenting an individual sentence pair (also known as online learning or adaptive machine translation).

1.1 Statistical Foundations

In this section, the foundations of statistical machine translation and its use in computer-aided applications are very briefly described.

1.1.1 Statistical Machine Translation

The statistical approach to MT formalises the problem of generating translations under a statistical point of view. More formally, given a source sentence $f_1^J \equiv f_1...f_j...f_J$ in the source language $\mathcal{F}$, we want to find its equivalent target sentence $e_1^I \equiv e_1...e_i...e_I$[a] in the target language $\mathcal{Y}$.

From the set of all possible sentences of the target language, we are interested in the one with the highest probability according to the following equation:

$$\hat{e}_1^{\hat{I}} = \arg\max_{I,e_1^I} \{Pr(e_1^I \mid f_1^J)\} \tag{1.1}$$

where $Pr(e_1^I \mid f_1^J)$ represents the probability of translating f_1^J into e_1^I.

The early works on SMT were based on the use of *generative models*. A generative model is a full probability model of all statistical variables that are required to randomly generating observable data. Generative models decompose $Pr(e_1^I \mid f_1^J)$ applying the Bayes decision

[a] f_j and e_i note the i'th word and the j'th word of the sentences f_1^J and e_1^I respectively.

rule. Taking into account that $Pr(f_1^J)$ does not depend on e_1^I we arrive to the following expression (Brown et al. 1993):

$$\hat{e}_1^{\hat{I}} = \arg\max_{I,e_1^I}\{Pr(e_1^I) \cdot Pr(f_1^J \mid e_1^I)\} \tag{1.2}$$

where: $Pr(e_1^I)$ represents the probability of generating the target sentence, and $Pr(f_1^J|e_1^I)$ is the probability of generating e_1^I given f_1^J . Since the real probability distributions $Pr(e_1^I)$ and $Pr(f_1^J|e_1^I)$ are not known, they are approximated by means of parametric statistical models. Specifically, $Pr(e_1^I)$ is modelled by means of a *language model*, and $Pr(f_1^J|e_1^I)$ is modelled by means of a *translation model*. Current MT systems are based on the use of *phrase-based models* (Koehn et al. 2003) as translation models. Typically, the values of the parameters of such statistical models are obtained by means of the well-known *maximum-likelihood* estimation method.

More recently, alternative formalizations have been proposed. Such formalizations are based on the direct modelling of the posterior probability $Pr(e_1^I|f_1^J)$, replacing the generative models by *discriminative models*. Log-linear models use a set of feature functions $h_m(f_1^J, e_1^I)$ each one with its corresponding weight λ_m:

$$\hat{e}_1^{\hat{I}} = \arg\max_{I,e_1^I}\left\{\sum_{m=1}^{M}\lambda_m h_m(f_1^J, e_1^I)\right\} \tag{1.3}$$

The direct optimization of the posterior probability in the Bayes decision rule is referred to as *discriminative training* (Ney 1995). Since the features of regular SMT log-linear models are usually implemented by means of generative models, discriminative training is applied here only to estimate the weights involved in the log-linear combination. This process is typically carried out by means of the *minimum error rate training* (MERT) algorithm (Och 2003).

1.1.2 Computer-Aided Translation

Despite multiple and important advances obtained so far in the field of SMT, current MT systems are in many cases not able to produce ready-to-use texts. Indeed, MT systems usually require human intervention in order to achieve high-quality translations. Here we consider two different types of computer-aided translation applications based on SMT: post-editing and interactive machine translation.

Post-Editing the Output of Statistical Machine Translation

Post-editing (PE) involves making corrections and amendments to machine generated translations (see (TAUS-Project 2010) for a detailed study). PE is used when raw machine translation is not error-free, situation which is common for current MT technology. PE started being used in the late seventies mainly at some big institutions (such as the European Commission) and is currently gaining acceptance from translation companies. Currently, PE tends to be carried out via tools built for editing human generated translations, such as translation memories (some authors refer to this task as simply *editing*). In addition to this, new translation memory tools and new versions of established ones offer translators the option to post-edit machine generated text for segments lacking any matches in the memories (Garcia 2011).

Since in the PE scenario, the user only edits the output of the MT system without further intervention from the system, there are no differences in the way in which the MT system is designed and implemented. Hence, the statistical framework for MT described above can be adopted without modifications in order to build the PE system.

Statistical Interactive Machine Translation

The interactive machine translation (IMT) framework constitutes an alternative to fully automatic MT systems in which the MT system and its user collaborate to generate correct translations. These correct translations are generated in a series of interactions between the ITP system and its user. Specifically, at each interaction of the ITP process, the ITP system generates a translation of the source sentence which can be partially or completely accepted and corrected by the user of the ITP system. Each partially corrected text segment (referred to from now on as prefix), is then used by the SMT system as additional information to generate better translation suggestions.

An example of a typical ITP session is shown in Figure 1.1. In interaction-0, the system suggests a translation ($\boldsymbol{s}$). In interaction-1, the user moves the mouse to accept the prefix composed of the first eight characters "To view " ($\boldsymbol{p}$) and presses the $\boxed{\mathbf{a}}$ key (k), then the system suggests completing the sentence with "*list of resources*" (a new $\boldsymbol{s}$). Interactions 2 and 3 are similar. In the final interaction, the user completely accepts the current suggestion.

Figure 1.1: ITP session to translate a Spanish sentence into English.

source(f_1^J): Para ver la lista de recursos

reference($\hat{e}_1^{\hat{I}}$): To view a listing of resources

interaction-0	$\boldsymbol{p}$						
	$\boldsymbol{s}$	*To*	*view*	*the*	*resources*	*list*	
interaction-1	$\boldsymbol{p}$	To	view				
	k			$\boxed{\mathbf{a}}$			
	$\boldsymbol{s}$				*list*	*of*	*resources*
interaction-2	$\boldsymbol{p}$	To	view	a	list		
	k				$\boxed{\mathbf{i}}$		
	$\boldsymbol{s}$				*ng*	*resources*	
interaction-3	$\boldsymbol{p}$	To	view	a	listing		
	k					$\boxed{\mathbf{o}}$	
	$\boldsymbol{s}$					*f*	*resources*
acceptance	$\boldsymbol{p}$	To	view	a	listing	of	resources

Figure 1.2 shows a schematic view of these ideas. Here, f_1^J is the input sentence and e_1^I is the output derived by the ITP system from f_1^J. By observing f_1^J and e_1^I, the user interacts with the ITP system, validating prefixes and/or pressing keys (k) corresponding to the next correct character, until the desired output $\hat{e}_1^{\hat{I}}$ is produced. The models used by the ITP system are

obtained through a classical batch training process from a previously given training sequence of pairs $(\boldsymbol{f}_n, \boldsymbol{e}_n)$ from the task being considered.

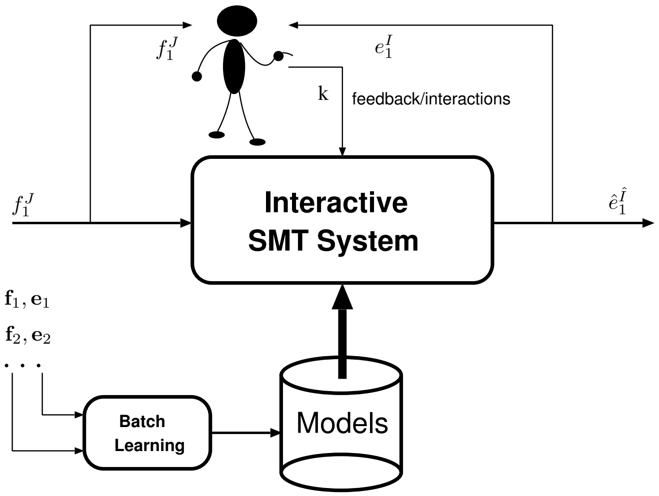

Figure 1.2: An Interactive SMT system.

More formally, in the ITP scenario we have to find an extension $\boldsymbol{s}$ for a prefix $\boldsymbol{p}$ given by the user:

$$\hat{\boldsymbol{s}} = \arg\max_{\boldsymbol{s}} \left\{ p(\boldsymbol{s} \mid f_1^J, \boldsymbol{p}) \right\} \tag{1.4}$$

Applying the Bayes rule, we arrive at the following expression:

$$\hat{\boldsymbol{s}} = \arg\max_{\boldsymbol{s}} \left\{ p(\boldsymbol{s} \mid \boldsymbol{p}) \cdot p(f_1^J \mid \boldsymbol{p}, \boldsymbol{s}) \right\} \tag{1.5}$$

where the term $p(\boldsymbol{p})$ has been dropped since it does not depend on $\boldsymbol{s}$.

Thus, the search is restricted to those sentences e_1^I which contain $\boldsymbol{p}$ as prefix. It is also worth mentioning that the similarities between Equation (1.5) and Equation (1.2) (note that $\boldsymbol{ps} \equiv e_1^I$) allow us to use the same models if the search procedures are adequately modified (Bender et al. 2005; Barrachina et al. 2009).

1.2 Toolkit Features

The toolkit includes the following features:

- Phrase-based statistical machine translation decoder.
- Computer-aided translation (post-edition and interactive machine translation).
- Incremental estimation of statistical models (adaptive machine translation).

- Client-server implementation of the translation functionality.
- Single word alignment model estimation using the incremental EM algorithm.
- Scalable and parallel model estimation algorithms using Map-Reduce.
- Compiles on Unix-like and Windows (using Cygwin) systems.
- Integration with the CasMaCat Workbench developed in the EU FP7 CasMaCat project[b].
- ...

1.3 Distribution Details

Thot has been coded using C, C++, Python and shell scripting. Thot is known to compile on Unix-like and Windows (using Cygwin) systems. As future work we plan to port the code to other platforms. See Section 1.6 section of this file if you experience problems during compilation.

It is released under the GNU Lesser General Public License (LGPL)[c].

1.4 Relation with Existing Software

Due to the strong focus of Thot on online and incremental learning, it includes its own programs to carry out language and translation model estimation. Specifically, Thot includes tools to work with n-gram language models based on incrementally updateable sufficient statistics. On the other hand, Thot also includes a set of tools and a whole software library to estimate IBM 1, IBM 2 and HMM-based word alignment models. The estimation process can be carried out using batch and incremental EM algorithms. This functionality is not based on the standard GIZA++ software for word alignment model generation.

Additionally, Thot does not use any code from other existing translation tools. In this regard, Thot tries to offer its own view of the process of statistical machine translation, with a strong focus on online learning and also incorporating interactive machine translation functionality. Another interesting feature of the toolkit is its stable and robust translation server.

1.5 Current Status

The Thot toolkit is under development. Original public versions of Thot date back to 2005 (Ortiz et al. 2005) and did only include estimation of phrase-based models. By contrast, current version offers several new features that had not been previously incorporated.

A basic usage manual is currently being developed. In addition to this, a set of specific tools to ease the process of making SMT experiments has been created.

In addition to the basic usage manual, there are some toolkit extensions that will be incorporated in the next months:

[b] http://www.casmacat.eu/
[c] http://www.gnu.org/copyleft/lgpl.html

- Improved management of concurrency in the Thot translation server (concurrent translation processes are currently handled with mutual exclusion) **[STATUS: implementation finished]**

- Virtualized language models, i.e. accessing language model parameters from disk **[STATUS: implementation finished]**

- Interpolation of language and translation models **[STATUS: implementation finished]**

Finally, here is a list of known issues with the Thot toolkit that are currently being addressed:

- Phrase model training is based on HMM-based alignments models estimated by means of incremental EM. This estimation process is computationally demanding and currently constitutes a bottleneck when training phrase models from large corpora. One already implemented solution is to carry out the estimation in multiple processors. Another solution is to replace HMM-based models by IBM 2 Models, which can be estimated very efficiently. However, we are also investigating alternative optimization techniques that allow us to efficiently execute the estimation process of HMM-based models in a single processor **[STATUS: under development, although code is much faster now]**

- Log-linear model weight adjustment is carried out by means of the downhill simplex algorithm, which is very slow. Downhill simplex will be replaced by a more efficient technique **[STATUS: issue solved]**

- Non-monotonic translation is not yet sufficiently tested, specially with complex corpora such as Europarl **[STATUS: under development]**

1.6 Documentation and Support

Project documentation is being developed. Such documentation include:

- Thot website[d].

- The Thot manual (`thot_manual.pdf` under the `doc` directory).

- Quick user guide[e].

- Seminar about statistical machine translation and Thot[f].

If you need additional help, you can:

- use the github issue tracker[g].

[d] http://daormar.github.io/thot/
[e] http://daormar.github.io/thot/docsupport/thot_quick_guide.pdf
[f] http://daormar.github.io/thot/docsupport/thot_seminar.pdf
[g] https://github.com/daormar/thot/issues

- send an e-mail to the author[h].
- join the CasMaCat support group[i].

Additional information about the theoretical foundations of Thot can be found in:

- Daniel Ortiz-Martínez. *Advances in Fully-Automatic and Interactive Phrase-Based Statistical Machine Translation*. *PhD Thesis*. Universitat Politècnica de València. Advisors: Ismael García Varea and Francisco Casacuberta. 2011.

One interesting feature of Thot incremental (or online) estimation of statistical models, is also described in the following paper:

- Daniel Ortiz-Martínez, Ismael García-Varea, Francisco Casacuberta. *Online learning for interactive statistical machine translation*. In Proc. of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT), pp. 546–554, Los Angeles, US, June 2010.

The phrase-level alignment generation functionality is described in:

- Daniel Ortiz-Martínez, Ismael García-Varea, Francisco Casacuberta. *Phrase-level alignment generation using a smoothed loglinear phrase-based statistical alignment model*. In Proc. of the European Association for Machine Translation (EAMT), pp. 160-169, Hamburg, Germany, 2008. *Best paper award*.

Finally, the initial version of Thot was described in:

- Daniel Ortiz-Martínez, Ismael García-Varea, Francisco Casacuberta. *Thot: a toolkit to train phrase-based models for statistical machine translation*. In Proc. of the Tenth Machine Translation Summit (MT-Summit), Phuket, Thailand, September 2005.

1.7 Citation

You are welcome to use the code under the terms of the license for research or commercial purposes, however please acknowledge its use with a citation:

- Daniel Ortiz-Martínez, Francisco Casacuberta. *The New Thot Toolkit for Fully Automatic and Interactive Statistical Machine Translation*. In Proc. of the 14th Annual Meeting of the European Association for Computational Linguistics (ACL): System Demonstrations, pp. 45–48, Gothenburg, Sweden, April 2014.

Here is a BiBTeX entry:

[h]daniel.o@webintepret.com
[i]http://groups.google.com/group/casmacat-support/boxsubscribe

```
@InProceedings{Ortiz2014,
  author    = {D. Ortiz-Mart\'{\i}nez and F. Casacuberta},
  title     = {The New Thot Toolkit for Fully Automatic and
               Interactive Statistical Machine Translation},
  booktitle = {14th Annual Meeting of the European Association for Computational
               Linguistics: System Demonstrations},
  year      = {2014},
  month     = {April},
  address   = {Gothenburg, Sweden},
  pages     = "45--48",
}
```

1.8 Sponsors

Thot has been supported by the European Union under the CasMaCat research project. Thot has also received support from the Spanish Government in a number of research projects, such as the MIPRCV project[j] that belongs to the CONSOLIDER programme[k].

[j] http://miprcv.iti.upv.es/
[k] http://www.ingenio2010.es/

Chapter 2

Installation

2.1 Basic Installation Procedure

The code of the Thot toolkit is hosted on github[a]. To install Thot, first you need to install the autotools (autoconf, autoconf-archive, automake and libtool packages in Ubuntu). If you are planning to use Thot on a Windows platform, you also need to install the Cygwin environment[b]. Alternatively, Thot can also be installed on Mac OS X systems using MacPorts[c].

On the other hand, Thot can be combined with third party software so as to enable extended functionality, see more information in Section 2.3.

Once the autotools are available (as well as other required software such as Cygwin, MacPorts), the user can proceed with the installation of Thot by following the next sequence of steps:

1. Obtain the package using git:

```
$ git clone https://github.com/daormar/thot.git
```

   Additionally, Thot can be downloaded in a zip file[d].

2. `cd` to the directory containing the package's source code and type `./reconf`.

3. Type `./configure` to configure the package.

4. Type `make` to compile the package.

5. Type `make install` to install the programs and any data files and documentation.

[a] https://github.com/daormar/thot/
[b] https://www.cygwin.com/
[c] https://www.macports.org/
[d] https://github.com/daormar/thot/archive/master.zip

6. You can remove the program binaries and object files from the source code directory by typing `make clean`.

By default the files are installed under the `/usr/local` directory (or similar, depending of the OS you use); however, since Step 5 requires root privileges, another directory can be specified during Step 3 by typing:

```
$ configure --prefix=<absolute-installation-path>
```

For example, if `user1` wants to install the Thot package in the directory `/home/user1/thot`, the sequence of commands to execute should be the following:

```
$ make clean # This is recommended if the package has already been built
$ ./reconf
$ configure --prefix=/home/user1/thot
$ make
$ make install
```

The installation process also creates three directories with additional information:

- **`${PREFIX}/share/thot/cfg_templates`**: contains configuration files to be used with different Thot utilities (see Chapter 3 for more details).
- **`${PREFIX}/share/thot/doc`**: contains the documentation of Thot, which currently consists in the Thot manual (`thot_manual.pdf`).
- **`${PREFIX}/share/thot/toy_corpus`**: contains a very small parallel corpus to make software tests. The directory includes both raw and preprocessed versions of the corpus (see Sections 3.3 and 3.4 for more details). This corpus may also be useful for new Thot users trying to get familiar with the toolkit functionality.

IMPORTANT NOTE: if Thot is being installed in a PBS cluster (a cluster providing `qsub` and other related tools), it is important that the `configure` script is executed in the main cluster node, so as to properly detect the cluster configuration (do not execute it in an interactive session).

2.2 Alternative Installation Options

The Thot `configure` script can be used to modify the toolkit behavior. Here is a list of current installation options:

- **--enable-ibm2-alig**: Thot currently uses HMM-based alignment models to obtain the word alignment matrices required for phrase model estimation. One alternative installation option allows to replace HMM-based alignment models by IBM 2 alignment models. IBM 2 alignment models can be estimated very efficiently without significantly affecting translation quality.

- **--with-kenlm=<DIR>**: installs Thot with the necessary code to combine it the with the KenLM library. `<DIR>` is the absolute path where the KenLM library was installed. See more information below.

- **--with-casmacat=<DIR>**: this option enables the configuration required for the CasMaCat Workbench. `<DIR>` is the absolute path where the CasMaCat library was installed. See more information below.

2.3 Third Party Software

Thot can currently be combined with third party software, extending its functionality. Below we enumerate the list of packages currently supported.

2.3.1 KenLM Library

The KenLM library[e] provides software to estimate, filter and query language models. KenLM has been incorporated into Thot so as to enable access of language model parameters from disk. The advantage of doing this is a reduction of main memory requirements and loading times for language models to virtually zero, at the cost of a small time overhead incurred by disk usage.

KenLM library should be downloaded, compiled and installed before executing Thot's `configure` script. `configure` should be used with the `--with-kenlm=<DIR>` option, where `<DIR>` is the directory where the library was installed. A specific version of KenLM has been created with minor modifications focused on making the package easier to install. The required sequences of commands is as follows:

```
$ mkdir kenlm ; cd kenlm
$ git clone https://github.com/daormar/kenlm.git repo
$ mkdir build ; cd build
$ cmake -DCMAKE_INSTALL_PREFIX=../ ../repo
$ make
$ make install    # Installs the library in the "kenlm" directory
```

For more information about how to use this functionality, please refer to Section 3.9.3.

[e] https://kheafield.com/code/kenlm/

2.3.2 Berkeley DB Library

Berkeley DB[f] is a software library providing a high performance database for key/value data. Berkeley DB can be combined with Thot so as to allow access to phrase model parameters from disk. The purpose of this is to reduce main memory requirements and loading times in the same way as was explained above for language models and KenLM.

Berkeley DB library should be installed in a standard OS directory (such as `/usr/local`) before the `configure` script is executed. In systems providing the `apt` tool, this can be easily achieved with the following command:

```
$ sudo apt install libdb++-dev
```

For additional information about how to use this functionality, see Section 3.9.4.

2.3.3 CasMaCat Workbench

Thot can be used in combination with the CasMaCat Workbench which has been developed in the project of the same name. The specific installation instructions can be obtained at the project website[g].

2.4 Checking Package Installation

Once the package has been installed, it is possible to perform basic checkings in an automatic manner so as to detect portability errors. For this purpose, the following command can be executed:

```
$ make installcheck
```

The tests performed by the previous command involve the execution of the main tasks present in a typical SMT pipeline, including training and tuning of model parameters as well as generating translations using the estimated models (see more on this in Section 3.1). The command internally uses the toy corpus provided with the Thot package (see Section 2.1) to carry out the checkings.

[f] http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html

[g] http://www.casmacat.eu/index.php?n=Workbench.Workbench

2.5 Add Thot to the System PATH

To end the installation process, it might be useful to add Thot to the system PATH. This will allow us to easily execute commands provided in the package without the necessity of providing the whole Thot installation path.

For this purpose, we can execute the following commands:

```
$ THOT_HOME_DIR=<absolute-installation-path>
$ export PATH=$PATH:${THOT_HOME_DIR}/bin
```

These variable definitions can be added to the `.bashrc` user profile file, so as to define them automatically whenever a new interactive shell session is started.

CHAPTER 3
USER GUIDE

This chapter provides usage information for the Thot toolkit. A toolkit overview is given in Section 3.1. A brief explanation about how corpora used in translation tasks are partitioned is provided in Section 3.2. The file naming conventions adopted to work with the toolkit are explained in Section 3.3. Corpus preprocessing functionality is shown in Section 3.4. Model training and tuning tools are presented in Section 3.5. Section 3.6 explains how to use the previously trained and tuned models to generate translations or phrase alignments. The tools useful to postprocessing the Thot's output are discussed in Section 3.7. Section 3.8 mentions some additional tools that are useful for translation tasks and Section 3.9 describes some advanced features implemented by Thot. Section 3.10 explains how Thot's configuration can be easily changed by means of the `master.ini` file. Section 3.11 provides general sample uses of Thot. Finally, Section 3.12 gives troubleshooting information about the toolkit usage.

3.1 Toolkit Overview

The basic usage of Thot involves training, tuning and search processes. Additionally, the parallel corpora used for training purposes can be previously preprocessed and the translation output may require a postprocess stage. The training process consists in estimating the parameters of the translation and language models. After that, a basic configuration file collecting the data of the trained models is generated (the workflow implemented by the Thot toolkit makes extensive use of configuration files to increase usability). Once these models have been generated, they are combined by means of the so-called log-linear models. This combination assigns different weights to the models so as to increase the translation quality. The exact values of such weights are determined during the tuning process. After training and tuning the models, an optional filtering step is carried out so as to keep the portion of the phrase model that is strictly necessary to work with a specific test corpus. This filtering process may be crucial to ensure the applicability of statistical machine translation in real scenarios, due to the huge size of the phrase models that are obtained when processing large training corpora. Finally, during the search process, the resulting model is applied to generate translations in a fully-automatic or interactive way, or to generate alignments at phrase level. Such processes can be summarized in the following list:

1. Corpus preprocessing
2. Language model training.
3. Translation model training.
4. Generate basic configuration file.
5. Parameter tuning.
6. Phrase model filtering (optional).
7. Search:
   (a) Fully automatic translation.
   (b) Interactive machine translation.
   (c) Phrase alignment generation.
8. Postprocessing of translator's output

Thot allows us to execute in parallel the above explained tasks using computer clusters or multiprocessor systems. Parallel implementation is transparent to the user, who is requested to specify the number of processors in which the tools will be executed. Thot currently supports the use of PBS clusters (a cluster providing `qsub` and other related tools).

In the following sections we describe the different tools offered by the Thot toolkit implementing the different steps of the translation pipeline described above.

3.2 Corpus Partition

SMT systems use parallel corpora to train and tune the model parameters. After the parameters have been estimated, the resulting models are used to obtain the target translations. The completion of these tasks require the generation of a corpus partition composed of three different sets:

- **Training set**: the training set is used to train the different statistical models involved in the translation. It is typically composed by many thousands or even millions of sentence pairs (greater training set sizes usually allow us to increase translation quality).
- **Development set**: the development set is used for parameter tuning (it is not used in the initial training stage). This set is typically composed of a few thousand sentence pairs (1 000 or 2 000 are usual in common translation tasks).
- **Test set**: the test set is used to compute automatic evaluation measures using the target sentence as reference sentences. This set is often composed of a few thousand sentence pairs in the same way as the development set.

In Thot it is assumed that, for a specific set, there will be two parallel files, one related to the source language and another related to the target sentence.

3.3 File Naming Conventions

To simplify the usage of some translation tools offered by Thot, it is useful to define a specific naming convention for the files containing the partition of the parallel corpus. In particular, the names will be composed by a prefix specific to the source or the target languages, and a suffix identifying whether the file belongs to the training, development or test set.

To illustrate this file naming convention, we can look at the files that compose the Spanish to English toy corpus included in the package (installed under the `${PREFIX}/share/thot/toy_corpus` directory, see Section 2.1):

- **{sp}|{en}.train**: the files `sp.train` and `en.train` compose the training set of the toy corpus, which is available in the Spanish and English languages.
- **{sp}|{en}.dev**: `sp.dev` and `en.dev` correspond to the development set.
- **{sp}|{en}.test**: finally, `sp.test` and `en.test` correspond to the test set.

As it is explained in the next section, the initial (or raw) corpus files can be preprocessed to improve translation results. When we carry out a specific preprocess step to a set of corpus files, we extend the prefix initially used to identify them. For instance, to identify the *tokenized* version of the previous toy corpus files (see Section 3.4.1), we add the string `_tok` to the prefix:

- **{sp_tok}|{en_tok}.train**
- **{sp_tok}|{en_tok}.dev**
- **{sp_tok}|{en_tok}.test**

After that, if the tokenized text is lowercased (see Section 3.4.2), the `_lc` string will be added as follows:

- **{sp_tok_lc}|{en_tok_lc}.train**
- **{sp_tok_lc}|{en_tok_lc}.dev**
- **{sp_tok_lc}|{en_tok_lc}.test**

Both the tokenized and lowercased versions of the toy corpus are included in the Thot package. Alternatively, they can be generated from the raw files using the corpus preprocessing tools described in the next section.

3.4 Corpus Preprocessing Tools

In common translation tasks, it is often interesting to preprocess the available parallel texts to make the translation process easier. Typically, corpus preprocessing involve three steps, namely, tokenization, lowercasing and corpus cleaning.

3.4.1 Tokenization

Before successfully aplying SMT, it is important to break the text into words, symbols or other meaningful elements that we will refer to as *tokens*. For this purpose, tokenization tools typically rely on a set of heuristics, such as discarding whitespace characters or splitting text by words and punctuation marks.

Thot provides the `thot_tokenize` tool, which can be used to tokenize texts. The tool receives the file to be tokenized using the `-f` parameter (or the standard input). This tool internally uses the above mentioned NLTK library, so it should be available before installing Thot (see Section 2.1).

Examples

The following list of commands obtain a tokenized version of the source files of the toy corpus:

```
thot_tokenize -f ${PREFIX}/share/thot/toy_corpus/sp.train > sp_tok.train
thot_tokenize -f ${PREFIX}/share/thot/toy_corpus/sp.dev > sp_tok.dev
thot_tokenize -f ${PREFIX}/share/thot/toy_corpus/sp.test > sp_tok.test
```

3.4.2 Lowercasing

After tokenizing the corpus, it can also be useful to lowercase it. This frees the translation system from the task of finding an appropriate capitalization for the translated text. In addition to this, the translations models will be hopefully improved, since now we are using a *canonic form* for each word, allowing us to better exploit the available training corpus.

To lowercase text, Thot provides the `thot_lowercase` tool. The tool receives the file to be lowercased using the `-f` option (or the standard input). This tool is also based on the NLTK library.

Examples

The following commands lowercase the tokenized source files of the toy corpus:

```
thot_lowercase -f ${PREFIX}/share/thot/toy_corpus/sp_tok.train >sp_tok_lc.train
thot_lowercase -f ${PREFIX}/share/thot/toy_corpus/sp_tok.dev > sp_tok_lc.dev
thot_lowercase -f ${PREFIX}/share/thot/toy_corpus/sp_tok.test > sp_tok_lc.test
```

3.4.3 Corpus Cleaning

The parallel texts used to train and tune the translation system can be cleaned by removing extremely long sentence pairs, which increase the parameter estimation time, as well as sentence pairs with highly disparate lengths, which may correspond to corpus segmentation errors. For this purpose, Thot incorporates the `thot_clean_corpus_ln` tool, whose main options are listed below:

thot_clean_corpus_ln	
-s ⟨string⟩	file with source sentences.
-t ⟨string⟩	file with target sentences.
-a ⟨int⟩	maximum sentence length allowed
-d ⟨int⟩	maximum number of standard deviations allowed in the difference in length between the source and target sentences.

The `thot_clean_corpus_ln` tool writes to the standard output the line numbers of all the sentence pairs of the given corpus that does not violate the length constraints mentioned above. On the other hand, it also print information about the line numbers of discarded sentence pairs to the error output. This information can be used to extract the corresponding source and target sentences using the `thot_extract_sents_by_ln` tool.

Examples

The following commands clean the tokenized and lowercased training set of the toy corpus:

```
thot_clean_corpus_ln -s ${PREFIX}/share/thot/toy_corpus/sp_tok_lc.train \
                     -t ${PREFIX}/share/thot/toy_corpus/en_tok_lc.train \
                     > line_numbers
thot_extract_sents_by_ln -f ${PREFIX}/share/thot/toy_corpus/sp_tok_lc.train \
                         -n line_numbers > sp_tok_lc_clean.train
thot_extract_sents_by_ln -f ${PREFIX}/share/thot/toy_corpus/en_tok_lc.train \
                         -n line_numbers > en_tok_lc_clean.train
```

3.5 Training and Tuning Tools

In this section, the functionality implemented in Thot for model training and tuning is described.

3.5.1 Language Model Training

Thot uses n-gram models with Jelinek-Mercer smoothing (see for instance (Chen and Goodman 1996)) to implement language models[a].

Basic Tools

Thot provides the `thot_lm_train` tool, which can be used to train language models. The basic input parameters include:

[a] State-of-the-art Kneser-Ney smoothing is currently being incorporated.

thot_lm_train	
-pr ⟨int⟩	number of processors used to perform the estimation.
-c ⟨string⟩	monolingual corpus used to estimate the model.
-o ⟨string⟩	directory for output files. When executing the command in PBS clusters, the user should ensure that the provided path is accessible for all nodes involved in the computation.
-n ⟨int⟩	order of the n-grams.

The command also generates a model descriptor in the output directory name `lm_desc` that will be useful to generate configuration files.

When executing the estimation process in computer clusters, it is important to ensure that the computer nodes involved receive enough resources (memory, computation time, etc.). For this purpose, the `thot_lm_train` tool incorporates de `-qs option`. In addition to this, `-tdir` and `-sdir` options allow to set the paths of the directory for temporary and shared files, respectively, when the default ones do not have enough free space to carry out the estimation.

For additional options and information, command help can be obtained by typing `thot_lm_train --help`.

Examples

To illustrate the tools related to language model generation, as well as for other examples shown in this chapter, we will use the toy corpus included with the Thot toolkit. Assuming that Thot was installed in `${PREFIX}` directory, the toy corpus will be available in `${PREFIX}/share/thot/toy_corpus`[b].

The following command line trains a 3-gram language model for the English tokenized and lowercased training set of the toy corpus, using the `-unk` option, storing the results in the `lm` directory:

```
train_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.train
thot_lm_train -c ${train_corpus} -o lm -n 3 -unk
```

3.5.2 Translation Model Training

Thot implements translation models by means of phrase-based models (Koehn et al. 2003).

Basic Tools

Thot incorporates the `thot_tm_train` tool, useful to train phrase models. The basic input parameters include:

[b] If `--prefix` option was not explicitly provided to `configure` during installation, then `${PREFIX}` is set to `/usr/local`.

thot_tm_train	
-pr ⟨int⟩	number of processors used to perform the estimation.
-s ⟨string⟩	file with source sentences.
-t ⟨string⟩	file with target sentences.
-o ⟨string⟩	directory for output files. When executing the tool in PBS clusters, the user should ensure that the provided path is accessible for all nodes involved in the computation.

The command also generates a model descriptor in the output directory name `tm_desc` that can be used to generate configuration files.

`thot_tm_train` also includes options to ensure that the training process receives enough computational resources in the same way as was explained for language model training (see Section 3.5.1). These options are `-qs`, `-tdir` and `-sdir`.

For additional options and information, command help can be obtained by typing `thot_tm_train --help`.

Examples

Again, we will use the toy corpus included with the Thot toolkit to illustrate the translation model functionality. Assuming that Thot was installed in `${PREFIX}` directory, the toy corpus will be available in `${PREFIX}/share/thot/toy_corpus`.

The following command line trains a phrase model for the tokenized and lowercased Spanish training set of the toy corpus, storing the results in the `tm` directory:

```
src_train_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.train
trg_train_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.train
thot_tm_train -s ${src_train_corpus} -t ${trg_train_corpus} -o tm
```

3.5.3 Basic Configuration File Generation

Thot uses configuration files to simplify the access to the toolkit functionality. These configuration files provides information about the parameters used by the Thot translation system, including the location of the translation and language models, the set of log-linear model weights, parameters relevant to the way in which the search process is carried out, etc.

Configuration files can be manually generated from the template files given in the path `${PREFIX}/share/thot/cfg_templates`, where `${PREFIX}` is the directory where Thot was installed.

Basic Tools

Alternatively, configuration files can also be generated by means of the `thot_gen_cfg_file` command. For this purpose, it is necessary to provide the descriptors of both the language and translation models (the exact syntax can be obtained by executing the command without parameters). The command output is written to the standard output and consists in a basic configuration file allowing to work with the language and translation models given by the provided model descriptors.

Examples

Assuming that we have already trained language and translation models located in the `lm` and `tm` directories, respectively, the following command line generates a basic Thot configuration file and writes it to the file `before_tuning.cfg`:

```
thot_gen_cfg_file lm/lm_desc tm/tm_desc > before_tuning.cfg
```

3.5.4 Parameter Tuning

After training the language and translation models, it is necessary to execute a parameter tuning stage. For this purpose, a development corpus separated from the training corpus is required. Currently, this stage affects to the weights of the language model as well as those of the log-linear model. Tuning of language model weights is necessary due to the use of Jelinek-Mercer smoothing (as it was mentioned above, Kneser-Ney smoothing is being implemented).

Thot incorporates the downhill-simplex algorithm (Nelder and Mead 1965) to tune the language and log-linear model weights. Regarding the criterion used for weight adjustment, language models weights are set so as to minimize the perplexity of the model, while the criterion to adjust log-linear weights is to maximize translation quality according to some specific measure (see Section 3.8.1 for additional details).

Basic Tools

The `thot_smt_tune` tool allows to perform parameter tuning. For this purpose, it is necessary a Thot configuration file and a development corpus. Here is a list of the basic input parameters:

thot_smt_tune	
`-pr ⟨int⟩`	number of processors used to perform the estimation.
`-c ⟨string⟩`	Thot configuration file.
`-s ⟨string⟩`	file with source sentences.
`-t ⟨string⟩`	file with target sentences.
`-o ⟨string⟩`	directory for output files. When executing the command in PBS clusters, the user should ensure that the provided path is accessible for all nodes involved in the computation.

`thot_smt_tune` returns a new configuration file where the language and log-linear model weights are tuned. This file is stored under the output directory given by the `-o` option with the name `tuned_for_dev.cfg`.

As in previously presented tools, `thot_smt_tune` may require specific resources that can be specified by means of the `-qs`, `-tdir` and `-sdir` options.

For additional options and information, command help can be obtained by typing `thot_smt_tune --help`.

Examples

The following command line tunes the system given in the `before_tuning.cfg` file, for the tokenized and lowercased development set of the Thot toy corpus, storing the results in the `smt_tune` directory:

```
src_dev_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.dev
trg_dev_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.dev
thot_smt_tune -c before_tuning.cfg -s ${src_dev_corpus} -t ${trg_dev_corpus} \
              -o smt_tune
```

After the successful execution of `thot_smt_tune`, the configuration file corresponding to the tuned system will be stored in `smt_tune/tuned_for_dev.cfg`.

3.5.5 Phrase Model Filtering

Phrase models are composed of millions of parameters when they are estimated from large training corpora, making impossible to store them in main memory when using regular computer hardware. One simple solution to this problem when the set of sentences to be translated is know beforehand is to filter those phrase model parameters that are relevant to carry out the translation process.

Basic Tools

The `thot_prepare_sys_for_test` tool allows to filter the parameters of a phrase model. For this purpose, it is necessary a Thot configuration file and a file with the sentences to be translated. Here is a list of the basic input parameters:

thot_prepare_sys_for_test	
-c ⟨string⟩	Thot configuration file.
-t ⟨string⟩	file with test sentences.
-o ⟨string⟩	directory for output files. When executing the command in PBS clusters, the user should ensure that the provided path is accessible for all nodes involved in the computation.

The `thot_prepare_sys_for_test` tool may require specific resources that can be specified by means of the `-qs`, `-tdir` and `-sdir` options, in a similar way to other tools described above.

For additional options and information, command help can be obtained by typing `thot_prepare_sys_for_test --help`.

Examples

The following command line filters the phrase model given in the `smt_tune/tuned_for_dev.cfg` file (obtained after the parameter tuning step), for the tokenized and lowercased test set of the Thot toy corpus, storing the results in the `filtered_models` directory:

```
src_test_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.test
thot_prepare_sys_for_test -c smt_tune/tuned_for_dev.cfg \
                          -t ${src_test_corpus} -o filtered_models
```

3.6 Search Tools

After performing model training, tuning and filtering, we are prepared to generate translations, both in a fully-automatic or interactive way. In addition to this, the estimated models can also be used to generate phrase level alignments. The following sections explain how to access such functionalities.

3.6.1 Fully Automatic Translation

Thot incorporates tools to translate a set of sentences in a fully-automatic way. This task can be carried out using both, command line and client-server tools.

Basic Tools

The `thot_decoder` tool allows to generate translations for a given test set. For this purpose, a Thot configuration file and a file with the sentences to be translated should be provided. Here is a list of the basic input parameters:

thot_decoder	
`-pr ⟨int⟩`	number of processors used to carry out the translation.
`-c ⟨string⟩`	Thot configuration file.
`-t ⟨string⟩`	file with test sentences.
`-o ⟨string⟩`	output file.

For additional options and information, command help can be obtained by typing `thot_decoder --help`.

Client-Server Tools

The translation functionality mentioned above is also included in the client-server architecture provided by Thot. This includes two basic tools: `thot_server` and `thot_client`.

The `thot_server` tool implements a fully-fledged SMT system. The most relevant input parameter that has to provided, the `-c` parameter, is the name of the configuration file.

On the other hand, `thot_client` can be used to request translations to the server. `thot_client` requires the IP address where the server is being executed using the `-i` option, as well as the sentence to be translated, that is provided by means of the `-t` option.

After finishing work with the server, we should not forget to end it. This can be done simply by killing the corresponding process or by using the `-e` option of `thot_client`.

Both the `thot_client` and `thot_server` tools may receive additional input parameters, use the `--help` option for more information.

Examples

The following command uses the system configuration provided in the `filtered_models/test_specific.cfg` file to translate the tokenized and lowercased test set of the Thot toy corpus, storing the results in the `output` file:

```
src_test_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.test
thot_decoder -c filtered_models/test_specific.cfg -t ${src_test_corpus} \
             -o output
```

Alternatively, the following example shows how to obtain the translation of a test sentence of the toy corpus using the client-server architecture:

```
thot_server -c smt_tune/tuned_for_dev.cfg & # start server
thot_client -i 127.0.0.1 -t 'voy a marcharme hoy por la tarde .'
```

After translating the previous sentence, and assuming there are no more to be translated, we can end the server using the following command:

```
thot_client -i 127.0.0.1 -e
```

IMPORTANT NOTES:

- It should be noted that in this example, we do not provide a filtered model but the general one (`smt_tune/tuned_for_dev.cfg`). This differs from the example given for the `thot_decoder` tool. In spite of the fact that filtered models can be loaded when using `thot_server`, the client-server implementation is more suited to work in a scenario where the sentences to be translated are not known beforehand and thus the models cannot be filtered.
- When using the client, it is strongly advised to type single instead double quotes to provide input strings. The reason why is that double quotes enable shell expansion (e.g. variable substitution), resulting in potentially undesired transformations in the strings. To avoid this, single quotes can be used.

3.6.2 Interactive Machine Translation

Thot implements interactive machine translation functionality, which allows to generate suffixes that complete the prefixes validated by the user (see Section 1.1.2).

Client-Server Tools

The interactive machine translation functionality implemented in Thot can be accessed by means of the `thot_server` and `thot_client` tools. This functionality includes obtaining the initial translation of a source sentence and obtaining the suffix that best completes a prefix given by the user.

Before using the interactive machine translation functionality, it is necessary to initialize the server. For this purpose, the `thot_server` tool requires the name of configuration file that is supplied by means of the `-c` parameter.

On the other hand, requests to the server can be sent by means of `thot_client`. As basic parameter, `thot_client` requires the IP address where the server is being executed using the `-i` option. Thot maintains state information through the different interactions between the user and the system. Some of the information that is mantained is specific to the user and hence it is important to use different user identifiers for different interactive translation sessions. The user identifier can be provided by means of the `-uid` option. To interactively translate a sentence, the `-sc` option is used to start the translation process. After that, new strings can be added to the previously existing prefix by means of the `-ap` option. It is important to point out that the server works at character level. Finally, the `-rp` option can be used to reset the prefix.

After finishing work with the server, it should be terminated. This can be done simply by killing the corresponding process or by using the `-e` option of `thot_client`.

To get additional information on the usage of `thot_client` and `thot_server`, use the `--help` option.

Examples

The following example shows how to access the basic interactive machine translation functionality provided by Thot using the client-server architecture:

```
thot_server -c smt_tune/tuned_for_dev.cfg & # start server
thot_client -i 127.0.0.1 -uid 0 -sc 'me marcho hoy por la tarde .'
thot_client -i 127.0.0.1 -uid 0 -ap 'I am '
thot_client -i 127.0.0.1 -uid 0 -ap 'le'
# NOTE: at this point, the user prefix provided to the system is 'I am le'
thot_client -i 127.0.0.1 -uid 0 -rp # reset prefix
```

After finishing the previous interactive machine translation session and provided that there are no additional sentences to be translated, the server can be terminated by typing:

```
thot_client -i 127.0.0.1 -e
```

3.6.3 Phrase Alignment Generation

TO-BE-DONE.

3.7 Output Postprocessing Tools

Once the output of the system have been obtained, it may be necessary to recase and/or detokenize the text depending on whether the initial corpus was preprocessed or not (see Section 3.4). Below we describe some tools that are helpful for this purpose.

3.7.1 Recasing

If the initial corpus was lowercased, it will be necessary to recase the output of the system. The Thot toolkit incorporates its own recasing tool: `thot_recase`. `thot_recase` works by estimating a recasing model from a raw text file. Once the model is estimated, it is applied to obtain the text in real case. The main options of the tool are the following:

thot_recase	
`-f` ⟨string⟩	file with text to be recased.
`-r` ⟨string⟩	file with raw text to train the recaser models.

One possible option to supply the raw text file would be simply passing the target training set. However, a better option could be to also include the target development set and the source test set as well (the model can take advantage of information in the source language for specific words). If the user adheres to the file naming conventions described in Section 3.3, then it is very easy to generate this raw text file by means of the `thot_gen_rtfile` tool, which requires the prefix of the source and target corpus files:

thot_gen_rtfile	
`-s` ⟨string⟩	prefix of files with source sentences.
`-t` ⟨string⟩	prefix of files with target sentences.

Examples

The following commands allows us to recase the output given by the decoder in the example shown in Section 3.6.1 (since the corpus was tokenized during the corpus preprocessing stage, we should provide the prefix corresponding to the tokenized files):

```
thot_gen_rtfile -s ${PREFIX}/share/thot/toy_corpus/sp_tok \
                -t ${PREFIX}/share/thot/toy_corpus/en_tok \
                > rtfile
thot_recase -f output -r rtfile > output_rec
```

3.7.2 Detokenization

The toolkit also incorporates a tool to detokenize texts: `thot_detokenize`. The tool works in a very similar manner to the `thot_recase` tool explained above. Specifically, it needs to train a detokenization model from raw text:

thot_detokenize	
-f ⟨string⟩	file with text to be detokenized.
-r ⟨string⟩	file with raw text to train the detokenizer models.

Again, the raw text file required by `thot_detokenize` can be generated by means of the `thot_gen_rtfile` tool, as it was explained above for the recaser.

Examples

The following commands allows us to detokenize the recased text obtained in the example of Section 3.6.1:

```
thot_gen_rtfile -s ${PREFIX}/share/thot/toy_corpus/sp \
                -t ${PREFIX}/share/thot/toy_corpus/en \
                > rtfile
thot_detokenize -f output_rec -r rtfile > output_rec_detok
```

3.8 Additional Tools

In the following sections we describe some tools relevant to the translation process that were not listed above.

3.8.1 Output Evaluation

After translating a test set, the translation quality can be evaluated using automatic measures provided that there exist reference translations for each source sentence. Thot implements some tools for this purpose:

- **thot_calc_bleu**: obtains the BLEU (bilingual evaluation understudy) measure (Papineni et al. 2001).

- **thot_calc_wer**: calculates the WER (word error rate) measure (the number of substitutions, insertions and deletions that are required to convert the system translation into the reference sentence).

3.8.2 Automatization of Translation Experiments

The `thot_auto_smt` tool allows users to conduct a whole SMT experiment in an automatic manner, including corpus preprocessing, parameter training and tuning, model filtering, generation of translations, output postprocessing and output evaluation. `thot_auto_smt` requires that the corpus files follow the naming conventions described in Section 3.3. Here is a list of the main options accepted by the tool:

thot_auto_smt	
`-pr ⟨int⟩`	number of processors used to carry out the translation.
`-s ⟨string⟩`	prefix of files with source sentences.
`-t ⟨string⟩`	prefix of files with target sentences.
`-o ⟨string⟩`	directory for output files. When executing the command in PBS clusters, the user should ensure that the provided path is accessible for all nodes involved in the computation.
`--skip-clean`	skip corpus cleaning stage.
`--tok`	execute corpus tokenizing stage.
`--lower`	execute lowercasing stage.
`--no-trans`	do not generate translations.

After completing its execution, the `thot_auto_smt` tool generates the following directories under the output directory provided as input parameter:

- **preproc_data**: contains the preprocessed corpus files (if preprocessing was requested).
- **lm**: contains the language model files.
- **tm**: contains the translation model files.
- **smt_tune**: contains all the data related to the tuned system.
- **filtered_models**: contains the data related to the system prepared for translation of the test set.
- **output**: contains the output of the system. Additionally, the tool also computes evaluation measures and postprocess the output if the corpus files were preprocessed.

3.9 Advanced Functionality

This section explains how to access advanced features included in the Thot toolkit.

3.9.1 Online Learning

Thot incorporates techniques that allow to incrementally update the parameters associated to the features of a state-of-the-art log-linear model (Ortiz-Martínez et al. 2010). For this purpose, a set of incrementally updateable sufficient statistics is defined for each feature, allowing us to process individual training samples in constant time complexity.

Client-Server Tools

The online learning functionality implemented in Thot can be accessed by means of the of the `thot_server` and `thot_client` tools. This functionality includes processing a single training pair and printing the models.

Prior to use the online learning functionality, the server should be initialized. For this purpose, the `thot_server` tool requires the name of configuration file that is supplied by means of the `-c` parameter.

On the other hand, requests to the server can be sent by means of `thot_client`. As basic parameter, `thot_client` requires the IP address where the server is being executed using the `-i` option. In addition to this, the `-tr` option can be used to specify the training pair to be processed and the `-pr` option allows to print the updated models to files. Printing the models causes the previous ones to be overwritten.

To get more information on the usage of `thot_client` and `thot_server`, use the `--help` option.

Examples

The following example shows how to process a new training pair and print the models using the client-server architecture:

```
thot_server -c smt_tune/tuned_for_dev.cfg & # start server
thot_client -i 127.0.0.1 -tr 'esto es una prueba' 'this is a test'
thot_client -i 127.0.0.1 -pr # print models (previous ones are overwritten)
```

3.9.2 Online Learning From Scratch

One interesting way to perform online learning arises when we start the learning process from scratch, i.e. with empty models. Thot incorporates one utility that automates this kind of initalization: `thot_sys_from_scratch`. This tool has only one mandatory option:

thot_sys_from_scratch	
`-o ⟨string⟩`	output directory.

`thot_sys_from_scratch` tool creates a directory containing empty language and phrase models, as well as a configuration file, `sys_from_scratch.cfg`, whose purpose is to be processed by `thot_server`. After initializing `thot_server` with the online learning from scratch configuration file, its models can be extended using `thot_client`.

Examples

The following example shows how to create an online learning system with empty models in the `ol_scratch` directory:

```
thot_sys_from_scratch -o ol_scratch
```

3.9.3 Accessing Language Model Parameters From Disk

Language models estimated from large corpora may have millions of parameters that should be loaded in memory so as to be accessed during the decoding process. As a consequence, the loading process is typically slow and requires a huge amount of main memory.

To avoid this problem, it is possible to handle language model parameters directly from disk. This reduces the main memory requirements and loading time costs to virtually zero.

Currently, Thot provides on-disk language model functionality by means of the KenLM library.

On-Disk Language Models Using KenLM

Enabling on-disk KenLM language models only requires the use of the `-kenlm` flag when executing the `thot_lm_train` utility. It is important to stress out that the `-kenlm` option will only be available if Thot has been properly installed with KenLM support (see Section 2.3.1).

As an example, the following command line trains a KenLM 3-gram language model for the English tokenized and lowercased training set of the toy corpus, storing the results in the `lm` directory:

```
train_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.train
thot_lm_train -c ${train_corpus} -o lm -n 3 -kenlm
```

It is important to remark here that KenLM language models are static. This means that they cannot be used for online learning.

3.9.4 Accessing Phrase Model Parameters From Disk

When estimated from large corpora, phrase models present the same problems that were explained above for language models, namely, increased main memory requirements and loading time costs.

One way to tackle this problem is, again, relying on disk to access model parameters. To offer this functionality, Thot uses the Berkeley DB library.

On-Disk Phrase Models Using Berkeley DB library

Berkeley DB library can be used to generate phrase models that are accessed from disk. For this purpose, the `-bdb` flag provided by the `thot_tm_train` tool should be applied. The flag will only be available if Thot was properly configured to work with Berkeley DB library (see Section 2.3.2).

The following command line trains a Berkeley DB phrase model for the tokenized and lowercased Spanish training set of the toy corpus, storing the results in the `tm` directory:

```
src_train_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.train
trg_train_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.train
thot_tm_train -s ${src_train_corpus} -t ${trg_train_corpus} -o tm -bdb
```

Phrase models based on the Berkeley DB library can be extended incrementally. However, extensions are time consuming, since their complexity depends on the number of parameters stored in the language model. This contrasts with the constant complexity offered by the native language model of Thot.

3.9.5 Using Multiple Language Models

In order to maximize translation quality, it can be interesting to maintain separated language models for data sets belonging to different domains. Once the models are estimated, the subsequent tuning process can be used to assign specific weights to each model according to their relative importance in the given translation task.

After training a language model using `thot_lm_train`, it is possible to add a new one by using the `-a` option provided by the same tool. For instance, assuming that the previously estimated language model was stored in the `lm` directory, the following command line adds a new trigram language model estimated from the training text pointed by the `train_corpus` shell variable:

```
thot_lm_train -c ${train_corpus} -a lm -n 3
```

The previous process can be repeated multiple times, using any of the options provided by `thot_lm_train`.

3.9.6 Using Multiple Phrase Models

Maintaining multiple phrase models can be interesting to maximize translation quality in the same way as was explained above for multiple language models.

The procedure to incorporate new phrase models after having trained the first one is also very similar to that followed for language models. In particular, the `-a` option for the `thot_tm_train` tool should be used. For example, the following command can be used to add a new phrase model to a previously estimated one stored in the `tm` directory (`src_train_corpus` and `trg_train_corpus` shell variables point to the source and target training text files, respectively).

```
thot_tm_train -s ${src_train_corpus} -t ${trg_train_corpus} -a tm
```

Again, the previous operation can be repeated multiple times using the different options provided by `thot_tm_train`.

3.9.7 Forcing Translations for Source Phrases

In certain translation scenarios, it may be important to force the translation of specific segments of the source sentence. For instance, forced translations may come from contextual terminologies, allowing system output to conform to the specific rules of the translation task.

Thot is able to fix the translations for source segments by incorporating specific XML tags in the sentences to be translated. Once these tags have been incorporated, the translation tools provided by Thot are used without modification.

The following input sentence in Spanish: "*buenos días*", incorporates XML tags indicating that the Spanish word *buenos* should be translated by the English word *good*.

```
<phr_pair_annot><src_segm>buenos</src_segm> <trg_segm>good</trg_segm></phr_pair_annot> días
```

Currently, this option is only available for the fully-automatic translation tools provided by Thot.

3.10 Changing Configuration Through master.ini File

Thot's code is structured in modules that are loaded dynamically. This allows to easily extend or modify the behavior of the toolkit. The modules loaded at a given instant are given as individual entries contained in a special file called `master.ini`. After installation, this file is located in the `${PREFIX}/share/thot/ini_files` directory.

Users can modify `master.ini` to enable certain functionality. In spite of the fact that `master.ini` can be modified directly, Thot provides a specific tool to manipulate this file: `thot_handle_ini_files`. We list the main options of the tool in the following table.

thot_handle_ini_files	
-s	show content of `master.ini`.
-w ⟨string⟩	overwrite current `master.ini` with the file provided as argument.
-r	revert `master.ini` to its original content after executing `make install`.

Thot currently provides a brief list of predefined `ini` files that will be extended in the future. After toolkit installation, such files are stored in the `${PREFIX}/share/thot/ini_files` directory:

- **`standard.ini`**: this `ini` file contains the standard or default configuration of the toolkit. `master.ini` file is initialized with this file every time `make install` is executed.
- **`ibm2.ini`**: this file replaces HMM-based alignment models by IBM 2 alignment models, which are used to smooth phrase model probabilities and also for phrase model estimation. IBM 2 alignment models substantially accelerate phrase model training with respect to HMM-based alignment models. If `configure` is executed with the `--enable-ibm2-alig` option, then `ibm2.ini` is used to initialize `master.ini` each time `make install` is executed.
- **`wer.ini`**: Thot uses the BLEU measure as the default translation quality measure. Such measure is used for instance to guide the tuning process. Thot offers the possibility to replace the BLEU measure by other measures by making the appropriate changes in the `master.ini` file. The `wer.ini` file replaces BLEU by the WER measure.

The details about how entries of `master.ini` should be defined and their relationship with the rest of the code and tools offered by Thot are provided in the code guide presented in Chapter 4.

3.11 General Sample Uses

In this section we will show some general sample uses illustrating the functionality of the Thot toolkit.

3.11.1 Training, Tuning and Translating

Example 1: processing the toy corpus

This example shows the sequence of commands required to train, tune and translate using the tokenized and lowercased version of the toy corpus included with Thot.

```
# define variables (optional)
src_train_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.train
trg_train_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.train
src_dev_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.dev
trg_dev_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.dev
src_test_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.test
trg_test_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.test

# train system
thot_lm_train -c ${trg_train_corpus} -o lm -n 3 -unk
thot_tm_train -s ${src_train_corpus} -t ${trg_train_corpus} -o tm

# generate cfg file
thot_gen_cfg_file lm/lm_desc tm/tm_desc > before_tuning.cfg
```

```
# tune system
thot_smt_tune -c before_tuning.cfg -s ${src_dev_corpus} -t ${trg_dev_corpus} \
              -o smt_tune

# filter phrase model
thot_prepare_sys_for_test -c smt_tune/tuned_for_dev.cfg -t ${src_test_corpus} \
                          -o filtered_models

# translate test corpus
thot_decoder -c filtered_models/test_specific.cfg -t ${src_test_corpus} \
             -o output

# evaluate translation quality
thot_calc_bleu -r ${trg_test_corpus} -t output
```

3.11.2 Online Learning

Example 1: adding a training pair to the toy corpus

The following example shows the commands that have to be executed to add a new training pair to the toy corpus. For this purpose, models for such corpus are first trained, tuned and filtered. After that, the Thot server is started and the client is used to incorporate the new training sample. Finally, the client is used again to print the resulting models to files, overwriting the previous ones.

```
# define variables (optional)
src_train_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.train
trg_train_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.train
src_dev_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.dev
trg_dev_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.dev
src_test_corpus=${PREFIX}/share/thot/toy_corpus/sp_tok_lc.test
trg_test_corpus=${PREFIX}/share/thot/toy_corpus/en_tok_lc.test

# train system
thot_lm_train -c ${trg_train_corpus} -o lm -n 3 -unk
thot_tm_train -s ${src_train_corpus} -t ${trg_train_corpus} -o tm

# generate cfg file
thot_gen_cfg_file lm/lm_desc tm/tm_desc > before_tuning.cfg

# tune system
thot_smt_tune -c before_tuning.cfg -s ${src_dev_corpus} -t ${trg_dev_corpus} \
              -o smt_tune

# filter phrase model
thot_prepare_sys_for_test -c smt_tune/tuned_for_dev.cfg -t ${src_test_corpus} \
                          -o filtered_models

# start Thot server
```

```
thot_server -c smt_tune/tuned_for_dev.cfg &

# add new training pair
thot_client -i 127.0.0.1 -tr 'esto es una prueba' 'this is a test'

# print models (warning: previously generated models are overwritten)
thot_client -i 127.0.0.1 -pr

# finish server
thot_client -i 127.0.0.1 -e
```

3.12 Troubleshooting

This section provides troubleshooting information about possible problems that arise during the toolkit usage. The current list of identified problems is the following:

- **The `thot_tm_train` tool is too slow**: Thot uses HMM-based alignment models to obtain the word alignment matrices required for phrase model estimation. The current implementation of this kind of models is slow and may constitute a bottleneck (the code will be optimized in future versions of Thot). To alleviate this problem, toolkit users may enable one of the following workarounds:

  a) Use the `-pr` option to execute `thot_tm_train` in multiple processors.

  b) Replace HMM-based alignment models by IBM 2 alignment models by means of the `--enable-ibm2-alig` option of the `configure` script (see Section 2.2 for more details). IBM 2 alignment models can be estimated and subsequentely used to generate word alignment matrices very efficiently without causing significant degradations in the translation quality. The use of this solution requires building again the package. In addition to this, previously estimated HMM-based alignment models (if any) using the toolkit will not be valid for the alternative build of the package and the user would need to re-train them.

CHAPTER 4

CODE GUIDE

This chapter provides an overview of Thot's code, which is mainly written in C++ and shell scripting. Section 4.1 offers a brief explanation about how the toolkit is configured before installation. A description of the main base classes used throughout the code is presented in Section 4.2. Section 4.3 analyzes the role that `master.ini` file plays in Thot's architecture. Section 4.4 gives a detailed explanation of model descriptor files, which constitute a key aspect to guarantee toolkit modularity and extensibility. Section 4.5 presents an overview of the main binary programs provided by the package. Finally, a brief look to one class with pivotal importance when implementing the translation server offered by Thot is given in Section 4.6.

4.1 Brief Introduction to Autotools

The Thot package has adopted the so-called GNU Build System to handle its compilation, installation and checking processes. The GNU Build System, in its simplest scenario is what allow users to build packages by executing `./configure && make && make install`.

In order to simplify the implementation of the GNU Build System for a particular package, it is possible to use the Autotools[a]. Autotools provide a portable, complete and self-contained GNU Build System.

When Autotools are used, implementing Thot's build system needs the definition of the `configure.ac` file as well a set of `Makefile.am` files for the different subdirectories of the package. The whole building process for Thot requires the following command line: `./reconf && ./configure && make && make install`.

The `reconf` shell script is a tool specifically created for Thot that executes some checkings and internally calls the `autoreconf` command provided by Autotools. As a result, the `configure` script is generated from the `configure.ac` file. Additionally, `autoreconf` also generates `Makefile.in` files from the corresponding `Makefile.am` files.

[a] http://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html

Once the `Makefile.in` files and the `configure` script have been generated, the latter is executed so as to obtain the final `Makefile` files that are used to build the package.

The `configure` script is not only a way to create a portable set of `Makefile`'s, but also offers a mechanism to handle building options for a given package. Thot makes extensive use of this feature, as it is shown for instance in Section 2.2.

Here we have provided a very brief introduction to the use of Autotools, however, the whole picture is much more complex. Interested readers are encouraged to consult detailed sources of documentation[b] to better understand how Thot's building system works.

4.2 Main Base Classes Used in Thot

Thot's C++ code makes an extensive use of base classes providing well defined interfaces. The main purpose of this is to take advantage of subtype polymorphism to allow for alternative implementations of the different software components involved in the translation process, making the code substantially easier to extend.

4.2.1 Base Classes for Statistical Machine Translation

In this section we list those main base classes that are related to statistical machine translation tasks:

- **`BaseWordPenaltyModel`**: this class defines the interface that word penalty models should provide. This kind of models are used control the length of the output sentences generated by the translation system.
- **`BaseNgramLM`**: defines the basic interface that should be implemented by n-gram language model classes.
- **`BaseSwAligModel`**: this base class determines the interface that single word alignment models should follow. Single word alignment models are used in Thot for probability distribution smoothing during the translation process and also during phrase model training.
- **`BasePhraseModel`**: this class provides the interface required by classes implementing phrase models.
- **`BasePbTransModelFeature`**: defines the interface that log-linear model features should follow.
- **`BaseSmtModel`**: provides the basis to implement statistical machine translation models that are used to score translation hypotheses generated during the decoding process.
- **`BaseStackDecoder`**: defines the interface requirements of classes implementing stack decoding algorithms. Stack decoding algorithms are used to find the translation of highest probability for a given source sentence[c].

[b] A complete tutorial is available in https://www.lrde.epita.fr/~adl/dl/autotools.pdf

[c] For more information see http://daormar.github.io/thot/docsupport/thot_seminar.pdf

- **`BaseLogLinWeightUpdater`**: it is used as base class for implementing log-linear weight updating components.
- **`BaseScorer`**: defines the interface for classes used to assign translation quality scores to the translations generated by Thot.
- **`BaseTranslationConstraints`**: defines an interface for classes specifying constraints over the translation system output. One example of these constraints is forcing the translation of certain segments of the source sentences, as it was explained in Section 3.9.7.

4.2.2 Base Classes Specific for Interactive Machine Translation

Besides previously listed base classes for statistical machine translation, there are also some that are specific for interactive machine translation:

- **`BaseErrorCorrectionModel`**: provides the interface for classes implementing stochastic error correction models.
- **`BaseEcModelForNbUcat`**: this class provides the interface for a specific variety of error correction models used in interactive machine translation systems.
- **`BaseWgProcessorForAnlp`**: this class is the basis to implement a component useful in interactive machine translation. In particular, a word graph processor. A word graph is a compact way to store a very long set of possible translations for a given source sentence. Each translation is stored as a whole path in the word graph starting from the empty hypothesis and ending when all source words have been translated.
- **`BaseAssistedTrans`**: this class defines the interface to implement assisted translation systems useful to perform interactive machine translation.

4.3 The `master.ini` File

As it was already explained in Section 3.10, Thot uses a file called `master.ini` defining which software modules are dynamically loaded when certain programs are executed. In this context, a module is a UNIX shared library (files with `so` extension), providing a specific implementation of a Thot's base class. Currently, the base classes covered in `master.ini` include the list given in Section 4.2 with the exception of `BasePbTransModelFeature` and `BaseSmtModel`.

Using the `master.ini` file allows to easily modify the behavior of the toolkit by only editing a text file. In addition to this, new modules can be created and used without the necessity of recompiling the Thot's code (only implementing a class derived from one of the base classes mentioned above is required). Appendix A provides a list of the software modules already implemented in Thot.

`master.ini` file is a plain text file composed of a set of entries, one per line. Below we show an example of one of these entries:

```
BaseScorer ; /home/dortiz/smt/software/thot_refactor/lib/mira_bleu_factory.so ;
```

Each entry is composed by three fields separated by semicolons. The first field is the name of the base class that is being implemented. The second field specifies the shared library that should be loaded. Finally, the third field contains input parameters that are passed to the constructor of the loaded class when creating class instances. This last field can be left blank.

4.4 Model Descriptor Files

As it was explained in Section 3.5, training tools for language and translation models generate model descriptor files (`lm_desc` and `tm_desc`, respectively). Such descriptors are composed of one or more entries, each one describing the basic information of a specific model.

A model descriptor entry, no matter if it is a language or a translation model descriptor, is a line with three fields separated by blanks (symbol # can be used to introduce comments). The first field specifies the shared library implementing the model. Such shared library will be loaded dynamically during model initialization. The second field is the relative path to the model files (more specifically, the prefix of the files is provided). Finally, the third field is a label associated to the model.

The following is an example of a language model descriptor entry:

```
/home/dortiz/smt/software/thot/lib/incr_jel_mer_ngram_lm_factory.so main/trg.lm main
```

Model descriptor files play a very important role in toolkit extensibility, since they allow to easily switch between different model implementations at running time.

4.5 Main Binary Programs Provided by Thot

Thot's functionality is built around a set of programs written in C++ that can be used to execute basic statistical machine translation services, such as training a model or translating sentences. Typically, those programs are later used within shell scripts to provide high level services. For instance, basic training programs mentioned above are not prepared to be executed in multiple processors or to deal with very large training corpora. Instead, they can be internally used by a shell script implementing *MapReduce* techniques, enabling parallelism and processing corpora of an arbitrary size. Another important purpose of high level services is to automate tedious processes, such as executing the whole training and translation pipeline with one command, including pre/post-processing steps.

The following list enumerates the main C++ programs implemented in Thot:

- **`thot_gen_phr_model`**: this program generates a phrase model given alignment information coming from single word alignment models. The estimation process that is executed is standard for phrase models, so no class given in `master.ini` file is loaded.

- **thot_gen_sw_model**: this program estimates the parameters of single word alignment models given a parallel training corpus. The above mentioned alignment information that is required to estimate phrase models is obtained as a by-product of single word model training using this tool. `thot_gen_sw_model` reads the `master.ini` file and loads the module related to `BaseSwAligModel` class.

- **thot_ll_weight_upd_nblist**: this tool is used to tune log-linear model weights given a list with the n-best translations for a development set.

- **thot_lm_perp**: this tool allows to calculate language model perplexity. Perplexity is a measure of the quality of a given language model. This tool is also valuable to debug newly implemented language models without the necessity of executing translation related tools, which use additional components and are much more complex. This tool reads `master.ini` and loads the module corresponding to class `BaseNgramLM`.

- **thot_query_pm**: this simple program allows to make queries to modules implementing phrase models. The main purpose of this is module debugging. This program loads the module corresponding to `BasePhraseModel` class in `master.ini`.

- **thot_ms_dec**: implements a fully automatic translation tool. This programs reads `master.ini` and loads the modules related to all of the statistical machine translation base classes listed in Section 4.2.1.

- **thot_ms_alig**: this program is used to generate phrase level alignments between sentence pairs. It loads from `master.ini` the same modules read by `thot_ms_dec`.

- **thot_server**: this tool implements the Thot server. It offers an important part of the functionality provided by the package, including fully-automatic translation, interactive machine translation or online training. `thot_server` loads all of the modules given in `master.ini`.

- **thot_client**: it implements the Thot client used to communicate with `thot_server`. `thot_client` does not need to load any module given in `master.ini`.

- **thot_scorer**: this tool is used to evaluate the translation quality measure of the system output given the corresponding reference translations. The specific translation quality measure computed by this tool is the one provided by the module for class `BaseScorer` in `master.ini` file.

4.6 The `ThotDecoder` Class

In the previous section, it was mentioned that the `thot_server` tool offers an important part of Thot's functionality, ranging from online model training to fully automatic or interactive machine translation. The `ThotDecoder` class is particularly interesting to study the C++ code implemented by the toolkit, since `thot_server` is completely built upon such class.

The ThotDecoder class is designed to provide translation services to a set of users, taking the responsibility to handle all of the information related to them. This involves the definition of a memory ownership model as well as a mechanism to manage concurrent accesses to the different services by multiple users.

The functions provided in the interface of the ThotDecoder class can be classified in the following groups:

- **Initialization**: memory ownership model is initialized from class constructor. Additionally, there is a function to initialize statistical machine translation data from a cfg file in the same way as it is done for high level programs, such as the above mentioned thot_server.
- **User related**: includes functions to create all the necessary data for a new user accessing the class services and also to release such data.
- **Training**: includes functions for online training of language and translation models and also to tune the log-linear model weights.
- **Automatic translation**: in this group we can find a function to translate a source sentence and another one to generate a phrase level alignment for a sentence pair.
- **Interactive translation**: the main functions in this group includes one to start the interactive machine translation process and another one to add a string to current translation prefix (see Section 1.1.2 for more details).

CHAPTER 5
BACKGROUND

TO-BE-DONE

Bibliography

Barrachina, S. et al. (2009). “Statistical Approaches to Computer-Assisted Translation”. In: *Computational Linguistics* 35.1, pp. 3–28.

Bender, O., S. Hasan, D. Vilar, R. Zens, and H. Ney (2005). “Comparison of Generation Strategies for Interactive Machine Translation”. In: *Conference of the European Association for Machine Translation*. Budapest, Hungary, pp. 33–40.

Brown, P. F., S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer (1993). “The Mathematics of Statistical Machine Translation: Parameter Estimation”. In: *Computational Linguistics* 19.2, pp. 263–311.

Chen, S. F. and J. Goodman (1996). “An Empirical Study of Smoothing Techniques for Language Modeling”. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Ed. by Arivind Joshi and Martha Palmer. San Francisco: Morgan Kaufmann Publishers, pp. 310–318.

Garcia, I. (2011). “Translating by post-editing: is it the way forward?” In: *Machine Translation* 25.3, pp. 217–237. ISSN: 0922-6567.

Koehn, P., F. J. Och, and D. Marcu (2003). “Statistical Phrase-Based Translation”. In: *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference*. Edmonton, Canada, pp. 48–54.

Nelder, J. A. and R. Mead (1965). “A Simplex Method for Function Minimization”. In: *The Computer Journal* 7.4, pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

Ney, H. (1995). “On the Probabilistic-Interpretation of Neural-Network Classifiers and Discriminative Training Criteria”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.2, pp. 107–119.

Och, Franz Josef (2003). “Minimum error rate training in statistical machine translation”. In: *Proceedings of the 41th Annual Conference of the Associations for Computational Linguistics*. Sapporo, Japan, pp. 160–167.

Ortiz, D., I. García-Varea, and F. Casacuberta (2005). “Thot: a Toolkit To Train Phrase-based Statistical Translation Models”. In: *Proceedings of the Machine Translation Summit X*. Phuket, Thailand, pp. 141–148.

Ortiz-Martínez, D., I. García-Varea, and F. Casacuberta (2010). “Online Learning for Interactive Statistical Machine Translation”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*. Los Angeles, pp. 546–554.

Papineni, K. A., S. Roukos, T. Ward, and W. Zhu (2001). *Bleu: a Method for Automatic Evaluation of Machine Translation*. Tech. rep. RC22176 (W0109-022). Yorktown Heights, NY: IBM Research Division, Thomas J. Watson Research Center, 10 pages.

TAUS-Project (2010). *Postediting in Practice. A TAUS Report*. Tech. rep. TAUS – Enabling better translation. URL: https://www.taus.net/reports/postediting-in-practice.

Appendix A

List of Thot modules

TO-BE-DONE