UNIVERSITY OF SOUTHAMPTON

Distance phrase reordering for MOSES

User Manual and Code Guide

by

Yizhao Ni, Mahesan Niranjan, Craig Saunders and Sandor Szedmak

Technical Report

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

April 30, 2010

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

by Yizhao Ni, Mahesan Niranjan, Craig Saunders and Sandor Szedmak

We describe the implementation of a novel distance phrase reordering (DPR) model for a public domain statistical machine translation (SMT) system - MOSES[1]. The model mainly focuses on the application of machine learning (ML) techniques to a specific problem in machine translation: learning the grammatical rules and content dependent changes, which are simplified as phrase reorderings. This document serves two purposes: a user manual for the functions of the DPR model and a code guide for developers.

[1] http://www.statmt.org/moses/

Contents

Acknowledgements

The work was supported by the PASCAL Network, School of Electronics and Computer Science, University of Southampton, and the European Commission under the IST Project SMART (FP6-033917). Moreover, particularly thanks are owing to Assistant Prof. Philipp Koehn and Dr. Hieu Hoang from University of Edinburgh, who provided valuable suggestions during this circuitous process.

Chapter 1

Introduction

1.1 Distance phrase reordering

The distance phrase reordering (DPR) model mainly focuses on the application of machine learning (ML) techniques to a specific problem in machine translation: learning the grammatical rules and content dependent changes, which are simplified as phrase reorderings. It models the problem with a classification framework and aims at improving the fluency of machine translation. Different from the lexicalized reordering model used in MOSES (Koehn et al., 2005), this model considers the sentence context as well as the relationships between phrase movements, by means of a newly emerging structured learning paradigm. As observed by the authors, the DPR model works well on some language pairs that contain many differences in word ordering (e.g. Chinese-to-English).

This document does not describe in depth the underlying framework and the readers are referred to (Ni et al., 2009) for more details about the model.

1.2 Copyright announcement

If you use this software in your scientific work, please cite the work (Ni et al., 2009).

Chapter 2

User manual

The purpose of this chapter is to offer a step-by-step example of downloading, compiling, and constructing a DPR model and its related integrating framework (i.e. the MOSES decoder (Koehn et al., 2007) and the minimal error-rating training (MERT) (Och, 2003)).

2.1 Source code

The DPR model is integrated into MOSES as a feature function. Therefore, you also need a MOSES software package to run the program. A MOSES package including the DPR model is available at the following location (the additional metadata named *DPR_MOSES.zip*):

http://eprints.ecs.soton.ac.uk/20939/

Alternatively, the source code is also available via Subversion from Sourceforge, by executing the following commands

```
mkdir MOSES_tools
svn co https://mosesdecoder.svn.sourceforge.net/svnroot/mosesdecoder
/branches/DPR_MOSES MOSES_tools
```

This will copy all source code (MOSES with DPR) to your local machine (in the directory */MOSES_tools/*).

2.2 Compilation

To compile the MOSES system, the readers are referred to the MOSES user guide (Koehn and Hoang, 2009). Note that the directory created in this report (i.e. */MOSES_tools/*) is equivalent to the directory */tools/moses/* mentioned in (Koehn and Hoang, 2009)[1].

To compile the DPR model, you need to go to the directory */MOSES_tools/DPR_model/* and execute the following command

```
./makeFile
```

If the program is compiled successfully, it will generate three executables

- smt_mainProcess_configuration
- smt_mainProcess_construct_phraseDB
- smt_mainProcess_generatePhraseOption

2.3 How to use

The DPR package consists of two modules: a sample extraction module (*smt_mainProcess_construct_phraseDB*) and a DPR probability generation module (*smt_mainProcess_generatePhraseOption*). The former is used to extract all samples (phrase pairs) for training a DPR model, while the latter is then used to generate the DPR probabilities for different phrase pairs.

2.3.1 Training a MOSES system

Since the DPR model requires some outputs from MOSES, you need to train a MOSES system before training a DPR model. The MOSES user guide will help you to complete this step.

2.3.2 Prerequisite

The DPR model requires the following outputs from a MOSES system

- **The source/target word-class dictionary**. After training a MOSES system, two files, named *fr.vcb.classes* and *en.vcb.classes*, are located in a local directory */root_directory/corpus/*[2]. Alternatively, you can use *mkcls* to train more accurate

[1] Read the paragraph under Section "Get the Latest Moses Version" in (Koehn and Hoang, 2009).
[2] The *root_directory* is the directory defined by the option *-root-dir* when training a MOSES system.

word-class dictionaries (e.g. by increasing training rounds, using different number of word classes, etc).

- **The word alignment file**. A file named *aligned.grow-diag-final-and*,which is in the directory */root_directory/model/*.
- **The phrase table generated by MOSES**. A file named *phrase-table.gz* is located in the directory */root_directory/model/* and you need to unzip it before using it. Alternatively, to facilitate the processing time of DPR it is highly recommended to use a filtered phrase table. That is, use the MOSES script *filter-model-given-input.pl*[3] to filter the phrase table and use the filtered table instead.

2.3.3 Generating a parameter configuration file

To construct a DPR model, the first step is to generate a parameter configuration file by calling

```
./smt_mainProcess_configuration myConfigurationFile
```

A file named *myConfigurationFile* will then be created, which contains all the information needed for the rest of the process. You need to fill in all items listed below[4]:

General part:

1. **SourceCorpusFile** - the source corpus for the training (each line is a sentence).
2. **TargetCorpusFile** - the target corpus for the training (each line is a sentence).
3. **SourceWordClassFile** - the source word-class dictionary from MOSES or *mkcls* (i.e. *fr.vcb.classes*).
4. **TargetWordClassFile** - the target word-class dictionary from MOSES or mkcls (i.e. *en.vcb.classes*).

For extracting samples (phrase pairs) for the DPR model:

5. **alignmentFile** - the word alignment file generated by MOSES (e.g. *aligned.grow-diag-final-and*).
6. (output) **phraseTableFile** - the file containing all samples (phrase pairs) for the DPR model.

[3] See Part V "Filtering Test Data" in (Koehn and Hoang, 2009)
[4] Note that certain items have been assigned default values

7. **TestFileName** - only source phrases appearing in this file will be extracted from the training corpus and form the sample pool. In order to facilitate the training process, it is highly recommended to define this file as the combination of the develop and the test sets (i.e. a text that containing all source sentences from the develop and the test sets).

For generating the DPR probabilities:

8. **PhraseTranslationTable** - the phrase table generated by MOSES (i.e. unzipped *phrase-table.gz*). It is highly recommended to use the filtered phrase table.
9. **maxTranslations** - the maximum number of translations for a source phrase (default 100).
10. **tableFilterLabel** - 0: the MOSES phrase table has not been filtered; 1 (default): the MOSES phrase table has been filtered.
11. (output) **weightMatrixFile** - the filename of the DPR model.
12. **weightMatrixTrainLabel** - 0: if you do not need to train a DPR model (e.g. you have trained it before); 1 (default): train a DPR model.
13. (output) **phraseOptionFile** - a file stores the phrase options (phrase pairs) with their DPR probabilities for each sentence in **TestFile**. Line i contains the phrase options for sentence i. This file will then be used by a MOSES decoder.
14. **TestFile** - the file containing the source test sentences. The phrase options with their DPR probabilities will be generated for these sentences only.
15. **batchOutputLabel** - 0: collect phrase options for one sentence at a time (use less memory but very slow); 1 (default and recommended): collect phrase options for all sentences at a time (use large memory but very fast).

For the DPR parameter settings:

16. **maxPhraseLength** - the phrase pairs up to length *maxPhraseLength* (default 7) will be extracted.
17. **classSetup** - the class setup of the DPR model, currently, the model only support 3-class setup and 5-class setup. See (Ni et al., 2009) for details.
18. **distCut** - prune the phrase pairs whose reordering distances are longer than *distCut* (default 15). To avoid some alignment errors caused by *GIZA++*.
19. **maxNgramSize** - the maximum length of ngrams used in the ngram feature dictionary (usually choose 3 or 4, default 4).

20. **windowSize** - the window size around the source phrases (usually choose 3 or 4, default 3). See (Ni et al., 2009) for details.

21. **minPrune** - prune the ngram features that occur less than *minPrune* times (default 1). See (Ni et al., 2009) for details.

22. **minTrainingExample** - prune the source phrases that occur less than *minTrainingExample* times (default 10), because the discriminative model does not work well when the training size is too small.

23. **maxRound** - the maximum number of iterations (default 500). See (Ni et al., 2009) for details.

24. **step** - the step size (learning rate) of the DPR model (default 0.05). See (Ni et al., 2009) for details.

25. **eTol** - the error tolerance for training the DPR model (default 0.001). See (Ni et al., 2009) for details.

2.3.4 Generating training samples for the DPR model

After completing the configuration file. Generating training samples for the DPR model is rather easy, just execute the command

```
./smt_mainProcess_construct_phraseDB myConfigurationFile
```

It will generate the following files for training the DPR model:

- **SourceCorpusFile.tags** - the word-class tags for the source corpus (each line is a sentence).
- **TargetCorpusFile.tags** - the word-class tags for the target corpus (each line is a sentence).
- **SourceCorpusFile.ngramDict** - the ngram feature dictionary constructed using the source word corpus.
- **TargetCorpusFile.ngramDict** - the ngram feature dictionary constructed using the target word corpus.
- **SourceCorpusFile.tagsDict** - the ngram word-class dictionary constructed using the source word-class corpus.
- **TargetCorpusFile.tagsDict** - the ngram word-class dictionary constructed using the target word-class corpus.

- **phraseTableFile** - the file containing all extracted samples (phrase pairs) for training the DPR model.
- **phraseTableFile.featureRelabel** - the relabel dictionary for the ngram features.

2.3.5 Training the DPR model and generating DPR probabilities

The final step is to execute the command

```
./smt_mainProcess_generatePhraseOption myConfigurationFile
```

and the following files will be generated:

- **weightMatrixFile** - the DPR model.
- **weightMatrixFile.startPosition** - the start position of each sub-DPR model (one for each unique source phrase).
- **phraseOptionFile** - the phrase options (each line is a sentence) for the **TestFile** corpus.
- **phraseOptionFile.startPosition** - the start position of each line in **phraseOptionFile**.

The phrase option files (i.e. **phraseOptionFile** and **phraseOptionFile.startPosition**) will then be used by the MOSES decoder.

2.3.6 Integrating the DPR model into MOSES

To integrate the DPR model into MOSES, you need to use the MOSES software package we provided (as some MOSES source code has been modified, see Section 4.7). Meanwhile, the following lines should be added to the file */root_directory/model/moses.ini*.

```
[DPR-file]
/your_directory_to_phraseOptionFile/phraseOptionFile
[wDPR]
the weight for the DPR model (e.g. 0.5)
[class-DPR]
the class for the DPR model (choose 3 or 5 depending on the DPR model trained)
```

This tells the MOSES decoder where the DPR-probability file is and what is the weight for the DPR model.

2.3.7 Minimal error-rating training (MERT)

To use MERT, you need to use the MOSES scripts package we provided (as some source code of the scripts has been modified, see Section 4.7). The scripts package is in the directory */MOSES_tools/scripts/* and the command is

```
./your_directory_to_scripts/training/mert-moses.pl
./your_directory_to_source/your_source_file ./your_directory_to_target/your_target_file
./your_directory_to_moses/moses-cmd/src/moses
./your_directory_to_model/model/moses.ini −−working-dir
./your_working_directory/ −−rootdir ./your_directory_to_scripts/ −−decoder-flags "-v 0"
```

If you would like to **switch on/off** the DPR model or other reordering models, you can use the configurations *lambdas* and *activate*. For example do the following

```
./your_directory_to_scripts/training/mert-moses.pl
./your_directory_to_source/your_source_file ./your_directory_to_target/your_target_file
./your_directory_to_moses/moses-cmd/src/moses ./your_directory_to_model/model/moses.ini
−−working-dir ./your_working_directory/ −−rootdir ./your_directory_to_scripts/
−−decoder-flags "-v 0" −−lambdas="wDPR:0.5,0.1-1.5"
−−activate=d_1, lm, tm, w, wDPR
```

The command tells MERT that the initial weight for the DPR model is 0.5 (you can also define weights for other parameters such as "d", "lm", "tm" and "w") and the range of the weight is between 0.1 and 1.5. Meanwhile, there are 5 weights needed tuning: d_1 (i.e. the word distance-based reordering model), lm (the language model), tm (the phrase translation model), w (the word penalty) and wDPR (the DPR model).

2.3.8 Decoding

When you obtain the tuned parameters for the MOSES decoder, use the following command to decode the test sentences

```
./your_directory_to_moses/moses-cmd/src/moses
-config /your_directory_to_model/model/moses.ini
-input-file /your_directory_to_source/your_source_test
1> /your_directory_to_target/your_target_translation 2> /your_directory_to_log/log_file
```

The translations will be written in the file *your_target_translation* and a log file *log_file* will also be created.

Now, enjoy the **distance phrase reordering model**!

2.4 Trouble shooting

When you compile the files or execute the commands, you might meet the following errors:

- **Permission denied**. Make sure the file is executable, you can change the mode of the file by using *chmod*

```
chmod u+x your_file
```

- **/bin/sh: ./check-dependencies.pl: /usr/bin/perlM: bad interpreter: No such file or directory**. This is due to different coding of CR (carriage return) between Windows and Linux (Unix) and cause a problem to function *check-dependencies.pl* (in the directory */MOSES_tools/scripts/*). You can try the Perl function *delDots.pl* [5] to solve the problem. Just do the following:

```
perl delDots.pl check-dependencies.pl check-dependencies1.pl
delete check-dependencies.pl
mv check-dependencies1.pl check-dependencies.pl
```

- **ERROR: Cannot find mkcls, GIZA++, & snt2cooc.out in . Did you install this script using 'make release'? at ./moses-script/scripts-20100427-2119/training/train-factored-phrase-model.perl line 152.** This might happen when you use *train-factored-phrase-model.perl* to train a MOSES system. The solution is to search "my $BINDIR=" in *train-factored-phrase-model.perl* and modify the line as

```
my $BINDIR="your_directory_to_GIZA++"
```

[5] The file is in the directory */MOSES_tools/*.

Chapter 3

Preliminary results

We now test the new MT system (MOSES with DPR) on an MT task: French to English translation. The *EuroParl* corpus[1] (French–English) was used, from which we extracted sentence pairs where both sentences had between 1 and 100 words, and where the ratio of the lengths was no more than $2:1$. The training set had $50K$ sentences whilst the develop and the test sizes were fixed at $1K$ sentences.

For parameter tuning, minimum-error-rating training (MERT) (Och, 2003) was applied. Experiments were repeated three times to assess variance and the performance was evaluated by four standard MT measurements, namely word error rate (WER), BLEU, NIST and METEOR (see (Callison-Burch et al., 2007) for details).

Table 3.1 demonstrates the translation results. In most of the cases, importing a DPR model improved the translation quality, especially the METEOR score.

System	MT evaluations			
	BLEU [%]	WER [%]	NIST	METEOR [%]
MOSES+LR+WDR	26.1 ± 0.1	39.0 ± 0.4	6.67 ± 0.04	48.7 ± 0.3
MOSES+DPR+LR+WDR	$\mathbf{26.5} \pm 0.3$	39.0 ± 0.1	$\mathbf{6.68} \pm 0.04$	$\mathbf{50.9} \pm 0.2$
MOSES+DPR+WDR	26.3 ± 0.1	$\mathbf{38.9} \pm 0.3$	$\mathbf{6.68} \pm 0.04$	50.7 ± 0.1
MOSES+DPR	26.3 ± 0.1	39.1 ± 0.2	6.66 ± 0.04	50.8 ± 0.1

TABLE 3.1: Evaluations for MT experiments. Bold numbers refer to the best results.

[1]The corpus can be downloaded at http://www.statmt.org/europarl/.

Chapter 4

Code guide

This chapter gives an overview of the code. The DPR model is implemented using object-oriented principles, and the developers can gain a general idea of its class organisation from this chapter. All source code is in the directory */MOSES_tools/DPR_model* and each class, function library and main process contains a brief description on its members and functions at the beginning of its .h/.cpp file.

As mentioned in Chapter 2, the DPR package consists of two modules: a sample extraction module (*smt_mainProcess_construct_phraseDB*) and a DPR probability generation module (*smt_mainProcess_generatePhraseOption*). The relationships among classes, function libraries and main processes are illustrated in Figure 4.1 and Figure 4.2.

In the following, we provide a summary of the package framework:

- The main processes: *smt_mainProcess_construct_phraseReorderingDB.cpp* and *smt_mainProcess_generatePhraseOption.cpp*.
- Processing a sentence:
  - **sentenceArray.h/cpp**. Store the words (or word-class tags) for a sentence.
  - **wordClassDict.h/cpp**. Store the word-class label for each word.
  - **phraseNgramDict.h/cpp**. Store the word/word-class ngram features.
  - **alignArray.h/cpp**. Store the word alignments for each sentence pair.
- Constructing and processing a sample (phrase pair) pool:
  - **phraseConstructionFunction.h/cpp**. Contain functions to construct the sample (phrase pair) pool.
  - **corpusPhraseDB.h/cpp**. Store the (source) phrases that appear in the train/test corpus.

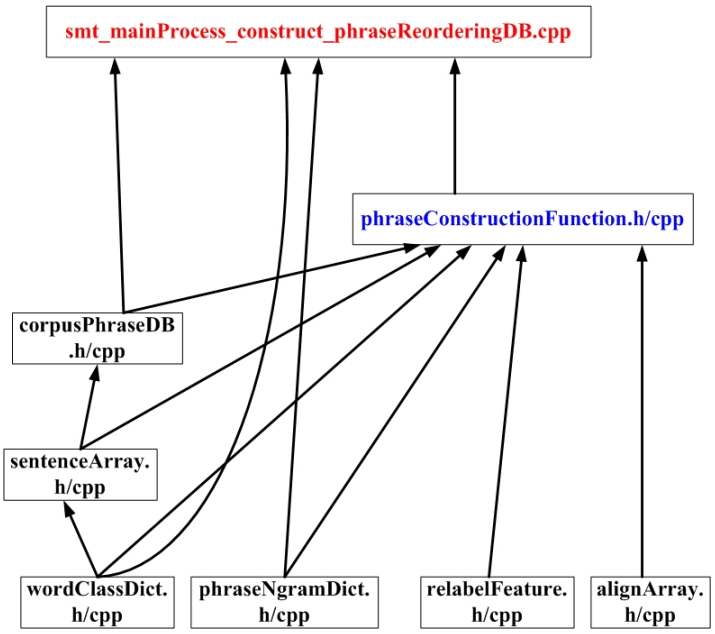

FIGURE 4.1: The relationships among classes, function libraries and main processes in the sample extraction module. The red block denotes the main process for this module (i.e. main.cpp); the blue block denotes the function library containing all functions needed in this module, and the black blocks are the classes. An arrow from block A to block B indicates that Block B directly calls functions (or uses classes) in Block A.

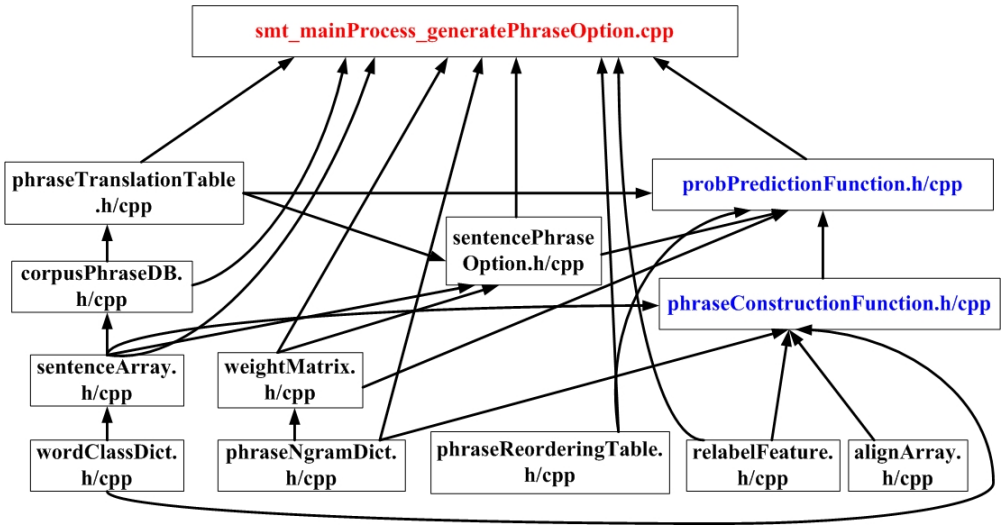

FIGURE 4.2: The relationships among classes, function libraries and main processes in the DPR probability generation module. The red block denotes the main process for this module (i.e. main.cpp); the blue block denotes the function library containing all functions needed in this module, and the black blocks are the classes. An arrow from block A to block B indicates that Block B directly calls functions (or uses classes) in Block A.

  - **phraseReorderingTable.h/cpp**. Store phrase pairs with their reordering distances (orientation class).
  - **phraseTranslationTable.h/cpp**. Store source phrases and their translations from a phrase table generated by Moses (to ensure the consistency between the two phrase pair databases).

- Constructing a DPR model:
  - **weightMatrix.h/cpp**. Train and store the weight matrix (matrices) of the DPR model.
  - **relabelFeature.h/cpp**. Store the relabel dictionary for ngram features (to reduce the size of the feature expression).

- Generating DPR probabilities:
  - **probPredictionFunction.h/cpp**. Contain functions to generate DPR probabilities for phrase options of each develop/test sentence.
  - **sentencePhraseOption.h/cpp**. Store phrase options (including target translations and DPR probabilities) for each develop/test sentence.

- The configuration process: **smt_configuration.cpp**.

- Other modifications on MOSES
  - **DPR_reordering.h/cpp**. An interface between the DPR model and the MOSES decoder.
  - **Parameter.cpp**
  - **StaticData.h/cpp**
  - **Makefile.am**
  - **mert-moses.pl**

The following sections specify the members and public functions for each class, function library and main process.

4.1 The main processes

Name	**smt_mainProcess_construct_phraseReorderingDB.cpp**
Function	Extract samples (phrase pairs) for training a DPR model
Inputs	
soucreCorpus (SourceCorpusFile)	the source corpus (text file)
targetCorpus (TargetCorpusFile)	the target corpus (text file)
wordAlignmentFile (alignmentFile)	the word alignment file (text file from GIZA++)
wordClassFile_fr (SourceWordClassFile)	the word-class dictionary for source words
wordClassFile_en (TargetWordClassFile)	the word-class dictionary for target words
maxNgramSize	the max length of ngram features
minPrune	prune ngram features that occur less than *minPrune* times
windowSize	the window size of the environment (for feature extraction)
maxPhraseLength	extract phrases upto length *maxPhraseLength*
testCorpusFile (TestFileName)	(optional) the source test corpus to filter the phrase DB
Outputs	
fout_phraseDB (phraseTableFile)	the output file of the phrase DB. Format: source phrase ||| target phrase ||| reordering dist ||| features
fout_relabelDB	the relabel dictionary of ngram features

Name	**smt_mainProcess_generatePhraseOption.cpp**
Function	A. Learn the sub-DPR model for each source cluster B. Construct the phrase option database
Inputs	
soucreCorpus (TestFile)	the source test corpus
sourceCorpus_tr (SourceCorpusFile)	the name of the source training corpus for reading word/word-class ngram dictionaries
targetCorpus_tr (TargetCorpusFile)	the name of the target training corpus for reading word/word-class ngram dictionaries
wordClassFile_fr (SourceWordClassFile)	the word-class dictionary for source words
wordClassFile_en (TargetWordClassFile)	the word-class dictionary for target words
extractPhraseTable (phraseTableFile)	the phrase pairs extracted for the DPR model
relabelDict	the relabel dictionary for ngram features
classSetup	current only support two class setups: 3 and 5
maxNgramSize	the max length of ngram features
minPrune	prune ngram features that occur less than *minPrune* times
windowSize	the window size of the environment (for feature extraction)
maxPhraseLength	extract phrases upto length *maxPhraseLength*
distCut	cut examples whose reordering distances are longer than *distCut*
maxRound	maximum iteration for training weight matrix W, see (Ni et al., 2009)
step	the learning rate of the PSL algorithm, see (Ni et al., 2009)
eTol	the error tolerance for training weight matrix W, see (Ni et al., 2009)
phraseTranslationTable	the phrase translation table from MOSES recommend using Moses's filtered translation table
filterLabel	1: the phrase translation table has been filtered 0: otherwise
batchLabel	1: store all sentence options first then output them at once, use large memory but fast 0: collect and output phrase options for one sentence at a time, use less memory but slower
maxTranslation	the maximum number of translation for each source phrase, if 0, use all translations
minTrainingExample	the minimum number of training examples required
Outputs	
fout_weightMatrix (weightMatrixFile)	the output file for the DPR model
fout_phraseOptionDB (phraseOptionFile)	the phrase option database for test sentences

4.2 Processing a sentence

Name	**sentenceArray.h/cpp**
Function	store the words for a sentence
Members	
sentence	(string array) store the words of a sentence
sentenceLengh	(int) store the sentence length
Public Functions	
sentenceArray()	constructor, create an empty sentence
sentenceArray(string sentenceString)	constructor, get words from a sentence string
sentenceArray(string sentenceString, wordClassDict* wordDict)	constructor, get the words and transform them to tags
string getPhraseFromSentence(int startPos, int endPos)	return the phrase sentence[startPos : endPos]
string getPhraseFromSentence(int startPos)	return the word sentence[startPos]
int getSentenceLength()	return the length of the sentence

Name	**wordClassDict.h/cpp**
Function	store the word-class label for each word
Members	
wordClassDictionary	(map), store the words (string) and the word-class labels (int)
readDictCheck	0: can not find the dictionary file 1: otherwise
numWords	the number of words in the dictionary
Public Functions	
wordClassDict(char* dictFileName)	constructor, read a dictionary file
bool checkReadFileStatus()	check the read status of the dictionary
void createWCFile(char* inputFile,char* outputFile)	output the dictionary to *outputFile* file
int getNumWords()	get the size of the dictionary
int getWordClass(string word)	get the word-class label of a word

Name	**phraseNgramDict.h/cpp**
Function	store the word/word-class ngram features
Members	
phraseDict	(map), store each phrase (ngram), its feature label, length and frequency
readDictCheck	0: can not find the dictionary file 1: otherwise
ngramIndex	the ngram label (used when constructing the dictionary)
Public Functions	
phraseNgramDict(char* dictFileName)	constructor, read a dictionary file
phraseNgramDict()	constructor, create an empty dictionary file
void insertNgram(string key, int ngramLength)	insert a new ngram feature
void deleteNgram(string key)	delete an ngram feature
int getNgramIndex(string key)	get the label of an ngram feature
int getNgramOccurance(string key)	get the frequency of an ngram feature
int getNgramLength(string key)	get the length of an ngram feature
vector<int> getNgramItems(string key)	get the (label, length, frequency) of an ngram feature
bool findNgram(string key)	search an ngram feature in the dictionary
bool checkReadFileStatus()	check the read status of a dictionary
void outputNgramDict(char* dictFileName, int minOccurenceCut)	output the dictionary to *dictFileName* file
int getNumFeature()	get the number of features in this dictionary

Name	**alignArray.h/cpp**
Function	store the word alignments for each sentence
Members	
align_FRtoEN	(map) the source to target alignment ([source word Pos]→[target word Pos])
align_ENtoFR	(map) the target to source alignment ([target word Pos]→[source word Pos])
Public Functions	
alignArray()	constructor, create an empty alignment file
alignArray(string alignmentString)	get the word alignments from a string
vector<int> getFRtoEN_alignment (int sourcePos)	return the corresponding target POSs for a source POS
vector<int> getENtoFR_alignment (int targetPos)	return the corresponding source POSs for a target POS
bool checkFRtoEN_alignment(int sourcePos)	check if the source POS is null aligned
bool checkENtoFR_alignment(int targetPos)	check if the target POS is null aligned

4.3 Constructing and processing a sample (phrase pair) pool

Name	**phraseConstructionFunction.h/cpp**
Function	contain functions to construct the sample pool
Public Functions	
bool smt_construct_phraseNgramDict(char* inputCorpusFile, char* ngramDictFile, int maxNgram, int minPrune)	construct the ngram dictionary for the source/target word/word-class tag corpus
phraseNgramDict smt_construct_phraseNgramDict(char* inputCorpusFile, char* ngramDictFile, int maxNgram, int minPrune, bool overloadFlag)	(overloaded) construct the ngram dictionary for the source/target word/word-class tag corpus
bool smt_construct_wordDict(char* wordClassDictFile, char* inputCorpus, char* tagsCorpus)	construct the word-class dictionary and create the tag corpus for the source/target corpus
wordClassDict smt_construct_wordDict(char* wordClassDictFile, char* inputCorpus, char* tagsCorpus, bool overloadFlag)	(overloaded) construct the word-class dictionary and create the tag corpus for the source/target corpus
vector<int> smt_extract_ngramFeature (sentenceArray* sentence, phraseNgramDict* ngramDictionary, int zoneL, int zoneR, int flag, int maxNgramSize)	extract ngram features around a source or target phrase
void smt_consistPhrasePair(sentenceArray* sentenceFR, sentenceArray* sentenceEN, sentenceArray* tagFR, sentenceArray* tagEN, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDictFR, phraseNgramDict* tagsDictEN, alignArray sentenceAlignment, int zoneConf[], int maxPhraseLength, int maxNgramSize, relabelFeature* featureRelabelDB, ofstream& fout)	extract all consistent phrase pairs (upto length *maxPhraseLength*) for a sentence pair using the word alignments (Time complexity $O(N^2)$)

Name	**phraseConstructionFunction.h/cpp**
Function	contain functions to construct the sample pool
Public Functions	Continued
void smt_consistPhrasePair(sentenceArray* sentenceFR, sentenceArray* sentenceEN, sentenceArray* tagFR, sentenceArray* tagEN, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDictFR, phraseNgramDict* tagsDictEN, alignArray sentenceAlignment, int zoneConf[], int maxPhraseLength, int maxNgramSize, relabelFeature* featureRelabelDB, ofstream& fout corpusPhraseDB* testPhraseDB)	(overloaded) extract all consistent phrase pairs (upto length *maxPhraseLength* and appeared in testPhraseDB) for a sentence pair using the word alignments (Time complexity $O(N^2)$)
void smt_constructPhraseReorderingDB(char* sourceCorpusFile, char* targetCorpusFile, char* wordAlignmentFile, char* tagsSourceFile, char* tagsTargetFile, char* phraseDBFile, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDictFR, phraseNgramDict* tagsDictEN, int zoneConf[], int maxPhraseLength, int maxNgramSize, char* featureRelabelDBFile)	extract all consistent phrase pairs with their reordering distances and ngram features (for all sentences in *sourceCorpusFile*)
void smt_constructPhraseReorderingDB(char* sourceCorpusFile, char* targetCorpusFile, char* wordAlignmentFile, char* tagsSourceFile, char* tagsTargetFile, char* phraseDBFile, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDictFR, phraseNgramDict* tagsDictEN, int zoneConf[], int maxPhraseLength, int maxNgramSize, char* featureRelabelDBFile, char* testFileName)	extract all consistent phrase pairs (appeared in *testFileName*) with their reordering distances and ngram features (for all sentences in *sourceCorpusFile*)

Name	**corpusPhraseDB.h/cpp**
Function	store the (source) phrases that appear in the train/test corpus
Members	
phraseDB	(map) store the phrases appeared in the corpus
numPhrase	(int) the number of phrases
maxPhraseLength	(int) the max phrase length in this phrase DB
Public Functions	
corpusPhraseDB()	constructor, create an empty phrase DB
corpusPhraseDB(char* inFileName, int MAXPLENGTH)	constructor, create a phrase DB for an input corpus
corpusPhraseDB(char* inFileName, int MAXPLENGTH, bool readDict)	constructor, read the phrase DB from a DB file
bool checkPhraseDB(string phrase)	check if a phrase appears in the phrase DB
int getNumPhrase()	return the number of phrases
int getMaxPhraseLength()	return the maximum phrase length
void outAllPhrases(char* outFileName)	output all phrases to *outFileName* file Format: phrase ||| phraseIndex

Name	**phraseReorderingTable.h/cpp**
Function	store the phrase pairs with their reordering distances (orientation class)
Members	
phraseTable	(map) store the source phrases with the orientation classes and the ngram features
numCluster	(int) the number of clusters (source phrases)
numPhrasePair	(int) the number of phrase pairs stored
positionIndex	(vector) store the start position of ngram features for each phrase pair in a position file
Public Functions	
sourceReorderingTable()	constructor, create an empty phrase table
sourceReorderingTable(char* inputFileName, int classSetup, int distCut)	constructor, read a phrase table from *inputFileName* file
createOrientationClass(int dist, int classSetup)	create the orientation class from the reordering distance of a phrase pair
int getClusterMember(string sourcePhrase)	get the number of examples in this cluster
vector<string> getClusterNames()	get all source phrases in the phrase table
int getNumCluster()	get the number of clusters (unique source phrases)
int getNumPhrasePair()	get the number of phrase pairs in the phrase table
int getNumOrientatin()	get the class setup
vector<vector<int>> getExamples(string sourcePhrase, ifstream& inputFile)	get the examples with their ngram features (store in a vector)
vector<unsigned long long> getPositionIndex()	get the start positions (in a position file) of ngram features for all phrase pairs

Name	**phraseTranslationTable.h/cpp**
Function	store source phrases and their translations from a phrase table generated by Moses
Members	
phraseTranslationTable	(map) a phrase table store source phrases → target translations
numCluster	(int) the number of clusters (unique source phrases) in the phrase table
numPhrasePair	(int) the number of phrase pairs in the phrase table
Public Functions	
phraseTranslationTable()	constructor, create an empty phrase table
phraseTranslationTable(char* inputFileName)	constructor, read the phrase pairs from an input file
phraseTranslationTable(char* inputFileName, int maxTranslations)	constructor, read the phrase pairs from an input file (for each phrase extract top *maxTranslations* translations)
phraseTranslationTable(char* inputFileName, corpusPhraseDB* testPhraseDB)	constructor, read the phrase pairs (appeared in *testPhraseDB*) from an input file
phraseTranslationTable(char* inputFileName, corpusPhraseDB* testPhraseDB, int maxTranslations)	constructor, read the phrase pairs (appeared in *testPhraseDB*) from an input file (for each phrase extract top *maxTranslations* translations)
vector<string> getClusterNames()	get all source phrases in the phrase table
int getNumCluster()	get the number of clusters (unique source phrases)
int getNumPhrasePair()	get the number of phrase pairs in the phrase table
vector<string> getTargetTranslation(string sourcePhrase)	get target translations for a source phrase
int getNumberofTargetTranslation(string sourcePhrase)	get the number of target translations for a source phrase

4.4 Constructing a DPR model

Name	**weightMatrix.h/cpp**
Function	train and store the weight matrix (matrices) of the DPR model (The file contains two classes)
Class	**weightMatrixW**
Members	
weightMatrix	(map) store the start positions of all sub-DPR models (one for each source phrase) in a weight matrix database
numCluster	(int) the number of clusters (source phrases)
Public Functions	
weightMatrixW()	constructor, create an empty dictionary. Format: source phrase → its DPR model in the database
weightMatrixW(char* inputFileName)	constructor, read the dictionary from an input file
int getNumCluster()	get the number of clusters (source phrases)
void writeWeightMatrix(char* outputFileName)	output the position dictionary to *outputFileName* file. Format: source phrase ||| start position
void insertWeightCluster(string sourcePhrase, unsigned long long startPos)	insert the start position of a new sub-DPR model
unsigned long long getWeightClusterPOS(string sourcePhrase)	get the start position of a sub-DPR model for a source phrase

Name	**weightMatrix.h/cpp** (continued)
Function	train and store the weight matrix (matrices) of the DPR model (The file contains two classes)
Class	**weightClusterW**
Members	
weightCluster	(map) store orientation → ngram features → feature values
numOrientation	(int) the number of orientation classes
sourcePhrase	(string) the source phrase for this sub-DPR model
distMatrix	(float matrix) the distance matrix (for structured learning)
Public Functions	
weightClusterW(string source, int numClass)	constructor, create an empty sub-DPR model (i.e. a weight cluster)
weightClusterW(ifstream& inputFile, int numClass, unsigned long long startPos)	constructor, read a sub-DPR model from an input file
int getNumOrientation()	get the number of classes
string getClusterName()	get the name (source phrase) of the sub-DPR model
unsigned long long writeWeightCluster(ofstream& outputFile)	output the sub-DPR model to *outputFile* file
void getWeightCluster(ifstream& inputFile, int numClass, unsigned long long startPos)	read the sub-DPR model from an input file
void structureLearningW(vector<vector<int>> phraseTable, int maxRound, float step, float eTol)	train a sub-DPR model using the structured learning algorithm
vector<float> structureLearningConfidence(vector<int> featureList)	return the confidence $W^T\phi(x)$ for each class
vector<float> structureLearningConfidence(vector<int> sourceFeature, vector<int> targetFeature)	(overloaded) return the confidence $W^T\phi(x)$ for each class

Name	**relabelFeature.h/cpp**
Function	Store the relabel dictionary for ngram features
Members	
featureRelabel	(map) the relabel dictionary of ngram features
countFeatureRelabel	(int) the number of features in the relabel dictionary
Public Functions	
relabelFeature()	constructor, create an empty relabel dictionary
relabelFeature(char* relabelFilename)	constructor, read a relabel dictionary from an input file
int insertFeature(int featureIndex)	relabel and insert an ngram feature
int getRelabeledFeature(int featureIndex)	return an ngram feature's relabeled feature
int getNumFeature()	return the number of relabeled features
void writeRelabelFeatures(char* dictFileName)	output the relabel dictionary to *dictFileName* file

4.5 Generating DPR probabilities

Name	**probPredictionFunction.h/cpp**
Function	contain functions to generate DPR probabilities for phrase options of each develop/test sentence
Public Functions	
void smt_sourceClusterPrediction(weightClusterW* wt, ifstream& sourceFeatureFileName, phraseFeaturePositionMap sourceFeaturePosition, targetFeatureMap targetTranslation, sentencePhraseOption* phraseOption)	for each cluster (source phrase), read its sub-DPR model W and predict the DPR probabilities (normalised) for each instance
void smt_sourceClusterPrediction(weightClusterW* wt, ifstream& sourceFeatureFile, phraseFeaturePositionMap sourceFeaturePosition, sourceTargetFeatureMapSTR::const_iterator sourceTargetFound, sentencePhraseOptionSTR *phraseOption)	(overloaded) for each cluster (source phrase), read its sub-DPR model W and predict the DPR probabilities (normalised) for each instance
void smt_createSourceCluster(char* inputFileName, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDict_fr, phraseNgramDict* tagsDict_en, wordClassDict* wordDict_fr, wordClassDict* wordDict_en, int maxPhraseLength, int maxNgramSize, int zoneConf[], relabelFeature* relabelDict, phraseReorderingTable* trainingPhraseTable, char* outputFileName, sourcePositionMap* sourcePositionDict)	given a test corpus, extract all source phrases appeared, store the source features in *outputFileName* file and return a *sourcePositionMap* dictionary

Name	**probPredictionFunction.h/cpp**
Function	contain functions to generate DPR probabilities for phrase options of each develop/test sentence
Public Functions	Continued
void smt_createSourceCluster(char* inputFileName, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDict_fr, phraseNgramDict* tagsDict_en, wordClassDict* wordDict_fr, wordClassDict* wordDict_en, int maxPhraseLength, int maxNgramSize, int zoneConf[], relabelFeature* relabelDict, phraseTranslationTable* trainingPhraseTable, char* outputFileName, sourcePositionMap* sourcePositionDict)	(overloaded) given a test corpus, extract all source phrases appeared, store the source features in *outputFileName* file and return a *sourcePositionMap* dictionary
void smt_createSourceCluster(string sourceSentence, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDict_fr, phraseNgramDict* tagsDict_en, wordClassDict* wordDict_fr, wordClassDict* wordDict_en, int maxPhraseLength, int maxNgramSize, int zoneConf[], relabelFeature* relabelDict, phraseTranslationTable* trainingPhraseTable, char* outputFileName, sourcePositionMap* sourcePositionDict)	(overloaded) given a test corpus, extract all source phrases appeared, store the source features in *outputFileName* file and return a *sourcePositionMap* dictionary

Name	**probPredictionFunction.h/cpp**
Function	contain functions to generate DPR probabilities for phrase options of each develop/test sentence
Public Functions	Continued
sentencePhraseOption smt_collectPhraseOptions(char* inputFileName, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDict_fr, phraseNgramDict* tagsDict_en, wordClassDict* wordDict_fr, wordClassDict* wordDict_en, int maxPhraseLength, int maxNgramSize, int zoneConf[], relabelFeature* relabelDict, phraseReorderingTable* trainingPhraseTable, char* weightFileName, weightMatrixW* weightMatrix	create phrase options for each test sentence Format: sentenceIndex→[left_boundary, right_boundary] →target translations → reordering probabilities
sentencePhraseOption smt_collectPhraseOptions(char* inputFileName, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDict_fr, phraseNgramDict* tagsDict_en, wordClassDict* wordDict_fr, wordClassDict* wordDict_en, int maxPhraseLength, int maxNgramSize, int zoneConf[], relabelFeature* relabelDict, phraseTranslationTable* trainingPhraseTable, char* weightFileName, weightMatrixW* weightMatrix, int classSetup)	(overloaded) create phrase options for each test sentence Format: sentenceIndex→[left_boundary, right_boundary] →target translations → reordering probabilities

Name	**probPredictionFunction.h/cpp**
Function	contain functions to generate DPR probabilities for phrase options of each develop/test sentence
Public Functions	Continued
void smt_collectPhraseOptions(char* inputFileName, phraseNgramDict* ngramDictFR, phraseNgramDict* ngramDictEN, phraseNgramDict* tagsDict_fr, phraseNgramDict* tagsDict_en, wordClassDict* wordDict_fr, wordClassDict* wordDict_en, int maxPhraseLength, int maxNgramSize, int zoneConf[], relabelFeature* relabelDict, phraseTranslationTable* trainingPhraseTable, char* weightFileName, weightMatrixW* weightMatrix, int classSetup, char* outPhraseOptionFileName)	(overloaded) create phrase options for each test sentence Format: sentenceIndex→[left_boundary, right_boundary] →target translations → reordering probabilities

Name	**sentencePhraseOption.h/cpp**
Function	store phrase options (including target translations and DPR probabilities) for each develop/test sentence (The file contains two classes)
Class	**sentencePhraseOption**
Members	
phraseOption	(map) store the phrase options. Format: sentenceID→[left_boundary, right_boundary]→ target translations (index)→reordering probabilities
numSen	(int) store the number of sentences
Public Functions	
sentencePhraseOption()	constructor, create an empty phrase option list
void createPhraseOption(int sentenceIndex, unsigned short phrase_boundary[], mapTargetProbOption targetProbs	compute the DPR probabilities for a phrase pair and update the phrase option list
void createPhraseOption(unsigned short phrase_boundary[], mapTargetProbOption targetProbs)	(overloaded) compute the DPR probabilities for a phrase pair and update the phrase option list
void outputPhraseOption(ofstream& outputFile, int sentenceIndex, sentenceArray* sentence, phraseTranslationTable* trainingPhraseTable)	output all phrase options to *outputFile* file
int getNumSentence()	get the number of sentences

Name	**sentencePhraseOption.h/cpp** (continued)
Function	store phrase options (including target translations and DPR probabilities) for each develop/test sentence (The file contains two classes)
Class	**sentencePhraseOptionSTR**
Members	also members inherited from **sentencePhraseOption**
phraseOption	(map) store the phrase options. Format: sentenceID→[left_boundary, right_boundary]→ target translations (index)→reordering probabilities
Public Functions	
sentencePhraseOptionSTR()	constructor, create an empty phrase option list
sentencePhraseOptionSTR(char* inputFileName)	constructor, read phrase options from inputFileName file
void outputPhraseOption(char* outputFileName)	output all phrase options to *outputFileName* file
void outputPhraseOption(ofstream& outputFile)	(overloaded) output all phrase options to *outputFileName* file
void createPhraseOption(int sentenceIndex, unsigned short phrase_boundary[], mapTargetProbOptionSTR targetProbs)	compute the DPR probabilities for a phrase pair and update the phrase option list
void createPhraseOption(unsigned short phrase_boundary[], mapTargetProbOptionSTR targetProbs)	(overloaded) compute the DPR probabilities for a phrase pair and update the phrase option list
vector<float> getPhraseProbs(int sentenceIndex, unsigned short phrase_boundary[], string targetPhrase, int numClass)	get the target translations and their DPR probabilities for a source phrase

4.6 The configuration process

Name	**smt_configuration.cpp**
Function	generate a configuration file for the DPR model

4.7 Other modifications on MOSES

To integrate the DPR model into the MOSES decoder, modifications are made to MOSES files **Parameter.cpp**, **StaticData.h/cpp** and **Makefile.am** in the directory

/MOSES_tools/moses/src/ and **mert-moses.pl** in the directory */MOSES_tools/scripts/training/*. To see these modifications, simply search "DPR" in the files.

A class **DPR_reordering.h/cpp** (in the directory */MOSES_tools/moses/src/*) is also created as an interface between the DPR model and the MOSES decoder.

Name	**DPR_reordering.h/cpp**
Function	an interface between the DPR model and the MOSES decoder
Members	
m_dprOptionStartPOS	(vector) store start positions of phrase options for each sentence (i.e. start positions of each line in the sentence option file)
sentenceOptionFile	(ifstream) the ifstream handle of the sentence option file
sentenceID	(long int) the test sentence ID
sentencePhraseOption	(map) store phrase options for each sentence
classSetup	(int) the number of orientations
unDetectProb	(float) the constant DPR probability for the phrase pair which is not in the sentence options
WDR_cost	(vector) the word distance-based reordering costs
Public Functions	
DPR_reordering(ScoreIndexManager &scoreIndexManager, const string filePath, const string classString, const vector<float>& weights)	constructor, read the sentence option file
size_t GetNumScoreComponents() const	return the number of score components (i.e. 1)
string GetScoreProducerDescription() const	return the name of the DPR model
string GetScoreProducerWeightShortName() const	return the short name of the weight for the DPR model (i.e. wDPR)
FFState* Evaluate(const Hypothesis& cur_hypo, const FFState* prev_state, ScoreComponentCollection* accumulator) const	compute DPR probabilities for the current extending phrase pair
const FFState* EmptyHypothesisState() const	initialisation function
void clearSentencePhraseOption() const	clear the phrase options in the option database
void constructSentencePhraseOption() const	construct phrase options for the current translating sentence
float generateReorderingProb(size_t boundary_left, size_t boundary_right, size_t prev_boundary_right, string targetPhrase) const	generate DPR probabilities for a phrase pair
int createOrientationClass(int dist) const	create the orientation class using the reordering distance

Bibliography

C. Callison-Burch, C. Fordyce, P. Koehn, C. Monz, and J. Schroeder. (meta-) evaluation of machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 136–158, Prague, Czech Republic, June 2007.

P. Koehn, A. Axelrod, A. B. Mayne, C. Callison-Burch, M. Osborne, and D. Talbot. Edinburgh system description for the 2005 iwslt speech translation evaluation. In *Proceedings of the International Workshop on Spoken Language Translation (IWSLT 2005)*, Pittsburgh, PA, October 2005.

P. Koehn and H. Hoang. Moses installation and training run-through. In *http://www.statmt.org/moses_steps.html*, December 2009.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico adn N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: open source toolkit for statistical machine translation. In *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session*, Prague, Czech Republic, 2007.

Y. Ni, C. Saunders, S. Szedmak, and M. Niranjan. Handling phrase reorderings for machine translation. In *Proceedings of the joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL–IJCNLP 2009)*, pages 241–244, Singapore, 2009.

F. J. Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, Japan, September 2003.