

PBML



The Prague Bulletin of Mathematical Linguistics

NUMBER 104 OCTOBER 2015

EDITORIAL BOARD

Editor-in-Chief

Jan Hajič

Editorial staff

Martin Popel
Ondřej Bojar

Editorial Assistant

Kateřina Bryanová

Editorial board

Nicoletta Calzolari, Pisa
Walther von Hahn, Hamburg
Jan Hajič, Prague
Eva Hajičová, Prague
Erhard Hinrichs, Tübingen
Aravind Joshi, Philadelphia
Philipp Koehn, Edinburgh
Jaroslav Peregrin, Prague
Patrice Pognan, Paris
Alexandr Rosen, Prague
Petr Sgall, Prague
Hans Uszkoreit, Saarbrücken

Published twice a year by Charles University in Prague

Editorial office and subscription inquiries:

ÚFAL MFF UK, Malostranské náměstí 25, 118 00, Prague 1, Czech Republic
E-mail: pbml@ufal.mff.cuni.cz

ISSN 0032-6585

PBML



The Prague Bulletin of Mathematical Linguistics

NUMBER 104 OCTOBER 2015

CONTENTS

Articles

| | |
|--|----|
| Joshua 6: A phrase-based and hierarchical statistical machine translation system | 5 |
| <i>Matt Post, Yuan Cao, Gaurav Kumar</i> | |
| Evaluating MT systems with BEER | 17 |
| <i>Miloš Stanojević, Khalil Sima'an</i> | |
| Box: Natural Language Processing Research Using Amazon Web Services | 27 |
| <i>Amittai Axelrod</i> | |
| Sampling Phrase Tables for the Moses Statistical Machine Translation System | 39 |
| <i>Ulrich Germann</i> | |
| Grasp: Randomised Semiring Parsing | 51 |
| <i>Wilker Aziz</i> | |
| MT-ComparEval: Graphical evaluation interface for Machine Translation development | 63 |
| <i>Ondřej Klejch, Eleftherios Avramidis, Aljoscha Burchardt, Martin Popel</i> | |
| TmTriangulate: A Tool for Phrase Table Triangulation | 75 |
| <i>Duc Tam Hoang, Ondřej Bojar</i> | |
| Instructions for Authors | 87 |



The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015 5-16

Joshua 6: A phrase-based and hierarchical statistical machine translation system

Matt Post^a, Yuan Cao^b, Gaurav Kumar^b

^a Human Language Technology Center of Excellence, Johns Hopkins University
^b Center for Language and Speech Processing, Johns Hopkins University

Abstract

We describe the version six release of Joshua, an open-source statistical machine translation toolkit. The main difference from release five is the introduction of a simple, unlexicalized, phrase-based stack decoder. This phrase-based decoder shares a hypergraph format with the syntax-based systems, permitting a tight coupling with the existing codebase of feature functions and hypergraph tools. Joshua 6 also includes a number of large-scale discriminative tuners and a simplified sparse feature function interface with reflection-based loading, which allows new features to be used by writing a single function. Finally, Joshua includes a number of simplifications and improvements focused on usability for both researchers and end-users, including the release of *language packs* — precompiled models that can be run as black boxes.

1. Introduction

Joshua¹ is an open-source toolkit for statistical machine translation of human languages. The Joshua 6 release introduces a phrase-based decoder that uses the standard priority-queue-based decoding algorithm (Koehn et al., 2003) to construct a hypergraph whose format is shared with the existing CKY+-based hierarchical decoding algorithms. This release also introduces a number of speed, memory, documentation, and infrastructure improvements designed to maximize usability in both research and production environments. This paper highlights these improvements and provides a

¹<http://joshua-decoder.org>

examples and usage notes for both the decoder and the Joshua pipeline, which takes care of all the steps of building and testing machine translation systems.

The original version of Joshua (Li et al., 2009) was a port from Python of the Hiero hierarchical machine translation system introduced by Chiang (2007). It was later extended (Li et al., 2010) to support grammars with rich syntactic labels, particularly “syntax-augmented” models (Zollmann and Venugopal, 2006). Subsequent versions produced Thrax, the extensible Hadoop-based grammar extraction tool for synchronous context-free grammars (Weese et al., 2011), later extended to support pivoting-based paraphrase extraction (Ganitkevitch et al., 2012). Joshua 5 (Post et al., 2013) introduced a sparse feature representation, support for GHKM (Galley et al., 2004, 2006) model construction, and large-scale discriminative tuners, as well as a number of significant improvements to speed and memory requirements.

2. Phrase-based decoder

The main feature of Joshua 6 is the introduction of a phrase-based decoder that is tightly integrated with the existing codebase. The phrase-based decoder is a variation of the classic priority-queue algorithm for phrase-decoding (Koehn et al., 2003). Briefly, the target-side sentence is built left-to-right, and the source sentence consumed in any order, subject to the distortion limit (controlled by the `-reordering-limit` flag, which defaults to 8). Joshua uses cube-pruning to moderate the search (Chiang, 2007; Huang and Chiang, 2007). Decoding iterates over stacks organized by the number of source words covered. A two-dimensional cube is constructed for each pairing of (a) a group of hypotheses from smaller stacks with identical coverage vectors and (b) the set of translations of a permissible source phrase extension of those hypotheses (with the number of translation options determined by `-num-translation-options`, defaulting to 20). Each cube is then added to a priority queue. Joshua iteratively consumes the top cube from the priority queue, extending the cube (a) to the next hypothesis with the same coverage vector and (b) to the next translation, and adding these extensions to the priority queue. Popping proceeds until the pop limit (`-pop-limit`, default 100) has been reached.

2.1. The hypergraph

Phrase-based decoding is typically presented as building a lattice, where nodes represent states (typically shared coverage vectors and target-side language model context) and arcs represent phrasal extensions. Conceptually, this is what Joshua does, but internally, it is using the same generalized hypergraph code used in the syntax-based decoder. To accomplish this, all phrases are read in as hierarchical rules with a single nonterminal on the left-hand side (essentially, phrases are reinterpreted as strictly left-branching grammar rules of arity 1). All applications of phrases must

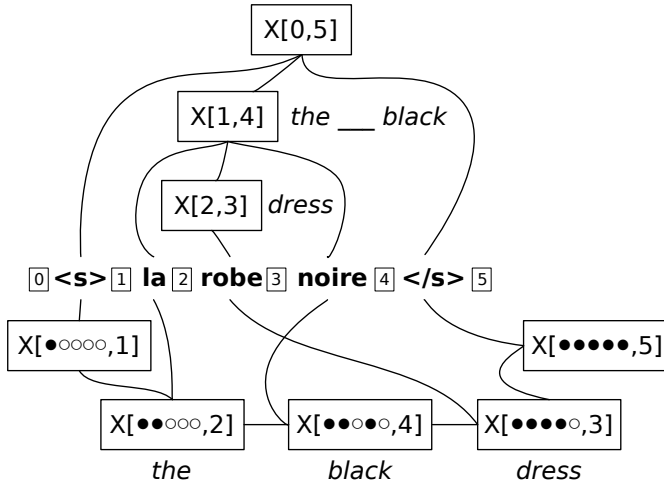


Figure 1. Shared hypergraph format when translating the French sentence *la robe noire* into English with the hierarchical (above) and phrase-based (below) decoders. The nodes are states in the hypergraph and contain the nonterminal label (here, *X*) and the input span (hierarchical) or coverage vector and last-translated word (phrase-based), as well as the target-side words produced by the incoming hyperedge.

extend an existing hypothesis, which is trivial since the stack decoding algorithm is seeded with an empty hypothesis representing the start of the sentence (Figure 1).

Sharing the hypergraph representation between the decoding algorithms provides many benefits. Feature functions can be written once and used for both decoders,² visualization tools work for both, and hypergraph operations such as minimum Bayes’ risk rescoring (Kumar and Byrne, 2004) work without modification.

2.2. Pipeline

Joshua’s `pipeline.pl` script can be invoked with a single command to run the entire process of building, tuning, and testing MT systems. The phrase-based decoder has been integrated, and can be enabled with the `--type {moses,phrase}` flag. The `moses` type uses Moses to build the phrase table, whereas `phrase` uses Joshua’s grammar extractor, Thrax (Weese et al., 2011). For example, the following command will do all of this for a Spanish–English Europarl system:

²This doesn’t preclude features that only make sense in one context; for example, the `Distortion` feature throws an error if its tail node can’t be recast as a phrase-based hypothesis with a coverage vector.

```
$JOSHUA/bin/pipeline.pl \
  --rmdir 1 --readme "Baseline phrase-based model" \
  --type phrase --source es --target en --corpus input/europarl-v7.es-en \
  --tune input/newstest2012 --test input/newstest2013 \
  --aligner berkeley --tuner mert --threads 2
```

There are many other options and intricacies to the pipeline; more information can be found with the Joshua documentation at <http://joshua-decoder.org/6.0/>.

2.3. Next Steps

Joshua’s phrase-based decoder is currently a “bare-bones” decoder, lacking state-of-the-art features such as lexicalized distortion and the operation sequence model (Durrani et al., 2011). We believe, however, that many of these gains can be implemented using the sparse feature framework (cf. Cherry (2013); Green et al. (2013)) rather than with hard-coded specialized modules.

We also plan to add a lattice decoding feature, which currently only works for the CKY+-based hierarchical system (where the implementation is simpler).

3. Feature function interface

Joshua’s *feature functions* are templates that contribute *features* to the global namespace. Whenever an edge is formed in the hypergraph, each feature function is asked to score it. During decoding, these are immediately scored against the weight vector to produce a scalar score; the individual feature values are then discarded, so as to avoid the overhead of storing the vectors. These values can be recovered later if desired (such as for parameter tuning) by replaying the feature functions.

Feature functions are written by extending the `FeatureFunction` class and overloading `compute(...)`. For example, the following `WordCounter` feature counts the number of times each target word is used:

```
package joshua.decoder.ff;

class WordCounter extends FeatureFunction {
  public DPState compute(Rule rule, List<HGNode> tails, int i, int j,
                        Sentence sentence, Accumulator acc) {

    for (int id: rule.getEnglish())
      if (id > 0) // skip nonterminals
        acc.add(String.format("WordCounter_%s", Vocab.word(id)), 1);

    return null;
  }
}
```


By convention, fired features are prefixed with the template name, so as to avoid clashes in the global namespace. The `Accumulator` object increments feature values and transparently handles either computing the feature dot product against the weight vector (during decoding) or retaining the actual feature values (during tuning). The `null` return value indicates that this function contributes no state.³

Features can be activated from the config file or command line:

```
$JOSHUA/bin/joshua-decoder -feature-function WordPenalty -key1 value ...
```

Joshua's features are loaded by reflection, so after compiling, there is no need to add stub code for recognizing and activating them. They also include a generic key-value argument-processing framework for passing parameters to the feature functions.

4. Class-based Language Models

Class based language models for machine translation (Wuebker et al., 2013) were proposed to combat data sparsity by building a language model over automatically-clustered words. The standard approach is to use a small number of classes (in the hundreds). This LM is generally used in addition to standard word-based LMs.

Joshua 6 allows the use of arbitrary word-classes for the purpose of class language model generation. The Joshua pipeline accepts a class map and proceeds to generate a class LM if this file exists.

```
$JOSHUA/bin/pipeline.pl [...] -class-map map.txt [...]
```

Class maps can be enabled in the decoder directly by passing the `-class-map` argument to the instantiation of a language model feature:

```
$JOSHUA/bin/joshua-decoder -feature-function 'LanguageModel \
  -path lm.kenlm -order 5 -class-map map.txt'
```

The class mapping file contains lines with a word followed by the class (space-delimited).

5. Parameter Tuning

Joshua 4 included the PRO tuner. Joshua 6 adds two new large-scale discriminative decoders: `k-best batch MIRA` (Crammer et al., 2006; Cherry and Foster, 2012) and `AdaGrad` (Duchi et al., 2011; Green et al., 2012). The usages of these tuners (as well as `Z-MERT`, which has always been a part of Joshua) are consistent, except for the class names and a few lines specifying the parameters in the configuration files.

A difficulty with decoding with large feature sets is that the set of observed features is not known prior to tuning. Joshua's discriminative tuners do not make any distinction between dense and sparse features, and will incorporate newly-fired features into their learning procedures, as those features are generated and encountered during the tuning process.

³e.g., language models are feature functions returning a state object representing the target-side context.

5.1. k-best batch MIRA

k-best batch MIRA is a variant of the “hope-fear” MIRA (Chiang et al., 2008) which uses k-best translations as approximate search spaces, and has been implemented in the Moses decoder (Cherry and Foster, 2012). In our implementation, in addition to the “hope-fear” pair (which balance the model and metric scores), we provide flexibility for also including the oracle (metric-best) and anti-oracle (metric-worst), similar to the hypothesis selection procedure proposed in Eidelman (2012). What is more, since MIRA is just like stochastic gradient descent (SGD) but with an adaptive learning rate, our implementation also allows using mini-batches for loss gradient estimation which reduces the estimation variance.

5.2. AdaGrad

AdaGrad is one of the best-performing online learning algorithms that has recently been applied to many NLP and deep learning tasks (Socher et al., 2013; Mnih and Kavukcuoglu, 2013; Chen and Manning, 2014) and to machine translation (Green et al., 2012). Our implementation includes the choice of using either L_1 and L_2 regularization. In the latter case, no closed-form solution to the update equation can be found (Duchi et al., 2011). However, we used a squared regularization term instead, which permits a closed-form update. We also use the structured hinge-loss as the objective, just like in the MIRA case, and mini-batch estimation of the gradient is also supported. Since when a large number of sparse features are defined, only a small part of them are active in each training sample, we use lazy update strategy in both the L_1 and L_2 regularization cases for those features that do not fire in each training sample.

6. Experiments

We present experiments on two language pairs: a hierarchical Chinese–English system, and a phrase-based Spanish–English system. The Chinese–English system was constructed from a variety of LDC resources, totaling just over 2M sentence pairs. The Hiero grammar was extracted with the default settings for Thrax, Joshua’s grammar extraction tool. A language model was built on Gigaword. We used the OpenMT 2012 data for tuning and evaluated against the NIST 2008 test set.

The Spanish–English system was built from Europarl using the `--type moses` flag to the Joshua pipeline. For tuning, we used the WMT 2012 news test set, and for testing, the 2013 one.

6.1. Phrase-based decoding

We compiled both Moses and mtplz (Heafield et al., 2014) with a number of optimizations (static linking, debug symbols off, max factors = 1, max kenlm order 5) and

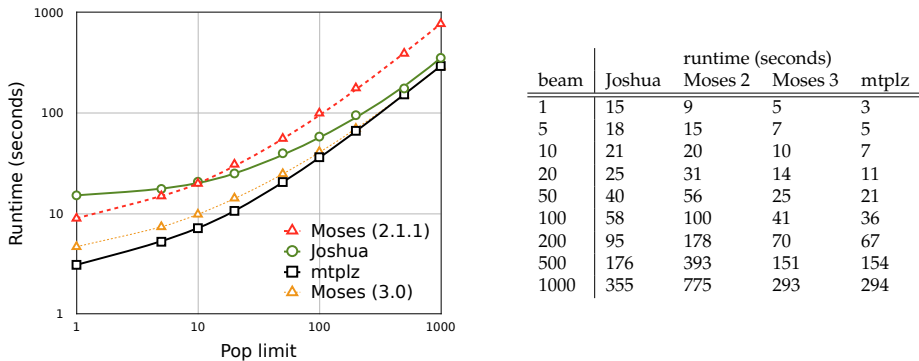


Figure 2. Decoding speeds for the 3,000-sentence newstest2013, varying the pop limits.

computed runtime for decoding all sentences in our ES-EN test set in single-threaded mode, not counting the model load time (except for mtplz, which includes it). Figure 2 plots run times as a function of the decoder pop limit. Joshua is mostly faster than Moses 2 except at the lowest two pop limits, but Moses 3 and mtplz are then about twice as fast again as Joshua at very low pop limits, which advantage disappears as the beam size is increased.

6.2. Parameter Estimation

We compare the performance of all the tuners implemented in Joshua (MERT, PRO, MIRA, AdaGrad) on the Spanish-English and Chinese-English systems. For each tuner, we repeated experiments five times with the training samples randomly shuffled. We compared systems with dense features only and dense+sparse features. The ten dense features are the regular MT features including phrase translation probabilities, the language model, word penalties, etc. The sparse features we use are the bigrams on the translation hypotheses. For the Spanish-English system, there are about 270k such features and for the Chinese-English system the number is about 60k. We ran each tuner 10 epochs on the tuning data set with a k-best list of size 300.

For PRO, we used the built-in binary Perceptron as the classifier. We sampled 8k training pairs from each k-best list and extracted the top 50 pairs to the classifier training set. For MIRA, the parameter C is set to 0.01, and we used mini-batch of size 10. For AdaGrad, we set $\lambda = 0.05$ and $\eta=0.1$ for both L_1 and L_2 regularizations, and also used mini-batch of size 10.

The experimental results on the test sets are given in Table 1. With only the small (dense) feature sets, all tuning algorithms in general give similar results, suggesting that they have probably found near-optimal solutions. When the bigram sparse fea-

| Tuner and Feature | Spanish–English | | Chinese–English | |
|---------------------|-----------------|-------|-----------------|-------|
| | Avg | Stdev | Avg | Stdev |
| MERT (dense) | 24.26 | 0.21 | 21.55 | 0.42 |
| PRO (dense) | 23.93 | 0.03 | 21.59 | 0.15 |
| PRO (dense+sparse) | 24.22 | 0.02 | 22.11 | 0.13 |
| MIRA (dense) | 24.22 | 0.05 | 21.81 | 0.07 |
| MIRA (dense+sparse) | 23.83 | 0.05 | 21.65 | 0.09 |
| AG-1 (dense) | 24.30 | 0.04 | 21.33 | 0.29 |
| AG-2 (dense) | 24.29 | 0.06 | 20.69 | 0.14 |
| AG-1 (dense+sparse) | 24.73 | 0.11 | 20.68 | 0.07 |
| AG-2 (dense+sparse) | 24.68 | 0.04 | 21.11 | 0.19 |

Table 1. A comparison of tuning algorithms implemented in Joshua. Here we show the average BLEU scores on the test set and their standard deviations of five repeated experiments on the Spanish–English and Chinese–English systems. AG-1, AG-2 means AdaGrad with L_1 and L_2 regularization respectively. The best average scores for each system are marked in bold.

tures are added, AdaGrad and PRO performed very well on the Spanish–English and Chinese–English systems, and yielded the best results. Although MIRA performed reasonably well when only dense features were present, it seems to suffer from overfitting when a large number of sparse features were added — we observed very good results on the tuning set but failed to see improvements on the test set. Finally, while AdaGrad gave the best results on the Spanish–English system, it did not perform as well on the Chinese–English system. Since AdaGrad makes use of the gradient information to scale the learning step in each dimension, it is very sensitive to magnitudes of gradient vectors (see the theoretical analysis in Duchi et al. (2011)). We therefore suspect that for the Chinese–English system, the loss gradients are very noisy and misguided AdaGrad to find inappropriate descent directions.

6.3. Class-based Language Models

We show results using Word2Vec (Mikolov et al., 2013) to generate word classes (though it would be just as easy to use Brown clusters (Brown et al., 1992; Liang, 2005) or any other deterministic mapping of words to classes). The word vectors were trained on the train partitions of each dataset. Results can be found in Table 2. We experimented with word vectors of various dimensions. Using this language model in addition to the word-based language model provides a gain of +0.53 BLEU on the Spanish–English dataset, but no gain on the hierarchical Chinese–English system (which may require a greater number of classes or an alternate way of clustering words into classes).

| System | BLEU score | |
|-------------------|-----------------|-----------------|
| | Spanish-English | Chinese-English |
| Baseline (Phrase) | 23.83 | 20.47 |
| + ClassLM (50) | 24.08 | 19.95 |
| + ClassLM (100) | 24.36 | 18.81 |
| + ClassLM (200) | 24.35 | 19.27 |
| + ClassLM (300) | 24.20 | 17.94 |

Table 2. A comparison of phrase-based systems that use class-based language model with a baseline phrase-based system. Classes are generated by clustering word vectors obtained by using Word2Vec. We show results for word vectors of dimensions 50, 100, 200 and 300. Class-LMs provide a significant BLEU gain with the Spanish-English system.

7. Language Packs

Even with the single-command pipeline invocation provided with Joshua, there are many impediments to building machine translation systems: one must select and obtain a large enough parallel dataset for training and tuning, have access to sufficient computing resources, and must have some familiarity with the steps of the pipeline should problems arise. These and other factors make it difficult for end users to install their own machine translation systems, and inhibit the adoption of customized statistical MT systems as tools in larger applications.

For this reason, the Joshua developers have released “language packs”: tuned models for particular language pairs that can be downloaded and run in a black-box fashion.⁴ Language packs include a tuned Joshua configuration file, all reference model files (the language model and the grammar or phrase table) in their respective compact, binarized formats, and scripts to perform source-side normalization and tokenization consistent with those used during training. The user is responsible for sentence-level segmentation.

7.1. Building a Language Pack

Building a language pack is simple. Joshua provides a script, `run_bundler.py` whose most important inputs are (a) a tuned Joshua configuration file and (b) the unfiltered translation model. The bundler creates a new directory and copies the model files into it, “packing” the Joshua translation model into its efficient binarized format. It then also copies the preprocessing scripts and the config file, relativizing path names and updating them to point to the unfiltered, packed, translation model. Finally, a shell script is created that serves as the entry point to running the decoder. An

⁴Available at <http://joshua-decoder.org/language-packs/>

example usage follows, where a pipeline run has taken place in the current directory and is being bundled into the directory `language-pack`:

```
$JOSHUA/bin/run_bundler.py tune/joshua.config.final language-pack \
  --copy-config-options '-top-n 0 -output-format %s -mark-oovs false' \
  --pack-grammar model/phrase-table.gz
```

The `--copy-config-options` parameters allows the config file options to be overridden (the values listed are the defaults), and `--pack-grammar` points to the unfiltered phrase table and requests that it be packed.

7.2. Running Language Packs

Language packs are run by executing the script `language-pack/run-joshua.sh`, which is meant to be used in the standard unix pipe fashion, taking input on `STDIN` and writing it to `STDOUT`. It is important that the user take care to pass sentences one per line, and to normalize and tokenize the input appropriately. This is accomplished with the `prepare.sh` script in the language pack. An example invocation is:

```
cat zh.txt | language-pack/prepare.sh | language-pack/run-joshua.sh > en.txt
```

Because of the overhead in loading models, language packs can also be run in server mode:

```
language-pack/run-joshua.sh -server-port 5867
cat zh.txt | language-pack/prepare.sh | nc localhost 5867 > en.txt
```

8. Summary

Joshua 6 is the result of a significant research, engineering, and usability effort that we hope will be of service to the research and open-source communities. In addition to the user-focused releases available at joshua-decoder.org,⁵ we encourage developers to contribute to the Github-hosted project at github.com/joshua-decoder/joshua. Mailing lists, linked from the main Joshua page, are available for both.

Acknowledgments Joshua’s phrase-based stack decoding algorithm began as a port of Kenneth Heafield’s ‘`mtplz`’ at MT Marathon 2014 in Trento, Italy.

Bibliography

- Brown, Peter F., Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18:467–479, 1992.
- Chen, Danqi and Christopher Manning. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of EMNLP*, Doha, Qatar, October 2014. ACL.

⁵All the features discussed in this paper are available as of Joshua 6.0.5.

- Cherry, Colin. Improved Reordering for Phrase-Based Translation using Sparse Features. In *HLT-NAACL*, pages 22–31. Citeseer, 2013.
- Cherry, Colin and George Foster. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the 2012 Conference of the North American Chapter of the ACL: Human Language Technologies*, pages 427–436, Montréal, Canada, June 2012. ACL.
- Chiang, David. Hierarchical Phrase-Based Translation. *Computational Linguistics*, 33(2):201–228, 2007.
- Chiang, David, Yuval Marton, and Philip Resnik. Online Large-Margin Training of Syntactic and Structural Translation Features. In *Proceedings of EMNLP*, Waikiki, Honolulu, Hawaii, October 2008. ACL.
- Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585, March 2006.
- Duchi, John, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, July 2011.
- Durrani, Nadir, Helmut Schmid, and Alexander Fraser. A Joint Sequence Translation Model with Integrated Reordering. In *Proceedings of the 49th Annual Meeting of the ACL: Human Language Technologies*, pages 1045–1054, Portland, Oregon, USA, June 2011. ACL.
- Eidelman, Vladimir. Optimization Strategies for Online Large-Margin Learning in Machine Translation. In *Proceedings of the 7th Workshop on Statistical Machine Translation*, Montréal, Canada, June 2012. ACL.
- Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. What’s in a translation rule? In *Proceedings of NAACL*, Boston, Massachusetts, USA, May 2004.
- Galley, Michel, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. Scalable Inference and Training of Context-Rich Syntactic Translation Models. In *Proceedings of ACL*, Sydney, Australia, July 2006.
- Ganitkevitch, Juri, Yuan Cao, Jonathan Weese, Matt Post, and Chris Callison-Burch. Joshua 4.0: Packing, PRO, and Paraphrases. In *Proceedings of the 7th Workshop on Statistical Machine Translation*, Montréal, Canada, June 2012. ACL.
- Green, Spence, Sida Wang, Daniel Cer, and Christopher D. Manning. Fast and Adaptive Online Training of Feature-Rich Translation Models. In *Proceedings of ACL*, Sofia, Bulgaria, August 2012. ACL.
- Green, Spence, Sida Wang, Daniel Cer, and Christopher D. Manning. Fast and Adaptive Online Training of Feature-Rich Translation Models. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 311–321, Sofia, Bulgaria, August 2013. ACL.
- Heafield, Kenneth, Michael Kayser, and Christopher D. Manning. Faster Phrase-Based Decoding by Refining Feature State. In *Proceedings of the ACL*, Baltimore, MD, USA, June 2014.
- Huang, Liang and David Chiang. Faster algorithms for decoding with integrated language models. In *Proceedings of ACL*, 2007.
- Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of NAACL*, Edmonton, Alberta, Canada, May–June 2003.

- Kumar, Shankar and William Byrne. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of NAACL*, Boston, Massachusetts, USA, May 2004.
- Li, Zhifei, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren N.G. Thornton, Jonathan Weese, and Omar F. Zaidan. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, 2009.
- Li, Zhifei, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Ann Irvine, Sanjeev Khudanpur, Lane Schwartz, Wren N.G. Thornton, Ziyuan Wang, Jonathan Weese, and Omar F. Zaidan. Joshua 2.0: a toolkit for parsing-based machine translation with syntax, semirings, discriminative training and other goodies. In *Proceedings of the Fifth Workshop on Statistical Machine Translation*, 2010.
- Liang, Percy. Semi-supervised learning for natural language. Master’s thesis, MIT, 2005.
- Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013.
- Mnih, Andriy and Koray Kavukcuoglu. Learning Word Embeddings Efficiently with Noise-Contrastive Estimation. In *Proceedings of NIPS*, Lake Tahoe, NV, USA, December 2013.
- Post, Matt, Juri Ganitkevitch, Luke Orland, Jonathan Weese, Yuan Cao, and Chris Callison-Burch. Joshua 5.0: Sparser, Better, Faster, Server. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 206–212, Sofia, Bulgaria, August 2013. ACL.
- Socher, Richard, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of EMNLP*, Seattle, USA, October 2013. ACL.
- Weese, Jonathan, Juri Ganitkevitch, Chris Callison-Burch, Matt Post, and Adam Lopez. Joshua 3.0: Syntax-based machine translation with the Thrax grammar extractor. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, 2011.
- Wuebker, Joern, Stephan Peitz, Felix Rietig, and Hermann Ney. Improving Statistical Machine Translation with Word Class Models. In *Conference on Empirical Methods in Natural Language Processing*, pages 1377–1381, Seattle, WA, USA, Oct. 2013.
- Zollmann, Andreas and Ashish Venugopal. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, New York, New York, USA, June 2006.

Address for correspondence:

Matt Post

post@cs.jhu.edu

Human Language Technology Center of Excellence, Johns Hopkins University
810 Wyman Park Drive, Baltimore, MD 21211



The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015 17-26

Evaluating MT systems with BEER

Miloš Stanojević, Khalil Sima'an

Institute for Logic, Language and Computation, University of Amsterdam

Abstract

We present BEER, an open source implementation of a machine translation evaluation metric. BEER is a metric trained for high correlation with human ranking by using learning-to-rank training methods. For evaluation of lexical accuracy it uses sub-word units (character n-grams) while for measuring word order it uses hierarchical representations based on PETs (permutation trees). During the last WMT metrics tasks, BEER has shown high correlation with human judgments both on the sentence and the corpus levels. In this paper we will show how BEER can be used for (i) full evaluation of MT output, (ii) isolated evaluation of word order and (iii) tuning MT systems.

1. Introduction

Machine Translation (MT) evaluation deals with the estimation of a measure (possibly distance) of the quality of some hypothesis MT output to given human translations, usually treated as gold standard translations. Often times, simplistic heuristics, such as counts of n-gram matches, are used. When the two corpora being compared are of relatively large size (>2000 sentences) the estimate can be reliable, even with simple measures such as BLEU (Papineni et al., 2002), because the collected sufficient statistic is reliable enough.

However, when we turn to evaluation at the sentence level, we cannot get away with such simple heuristic measures based on simple counting of too sparse statistics. We believe that evaluation should be treated as a modeling task (just like parsing or POS tagging or other NLP tasks) where we should train our models to learn the specific aspects of language processing, using a wide range of quality indicators (features). This is the motivation for making BEER a trained metric.

It is no surprise that trained metrics perform much better than heuristic metrics. BEER has been the best sentence level metric on WMT14 metrics task (Macháček and Bojar, 2014), and one of the best on both corpus and sentence level on WMT15 metrics task (Stanojević et al., 2015a). On the WMT15 tuning task for Czech-English BEER was the best submitted system falling behind only over the strong baseline (Stanojević et al., 2015b).

Unfortunately, trained metrics are often not that easy to use. Furthermore, these metrics are mostly made for the metrics tasks and often not published online, and when they are published online, this is done without the trained models or without suitable documentation. With this paper we aim to document BEER as a trained metric that performs well but also is easy to use, offering a range of attractive properties that researchers are used to see in the simple measures (for example statistical testing, tuning and many more).

In this paper we will concentrate only on the usage of BEER, but BEER has many interesting aspects that are presented elsewhere:

- evaluation of word order using permutation trees (PETs) (Stanojević and Sima'an, 2014b)
- character n-gram matching (Stanojević and Sima'an, 2014a)
- a learning-to-rank model (Stanojević and Sima'an, 2014a)
- a corpus level score that decomposes to sentence level scores (Stanojević and Sima'an, 2015)
- a tuning model that is not biased for recall (Stanojević and Sima'an, 2015)
- a Treepel version based on syntactic features (Stanojević and Sima'an, 2015)

In the next section we will briefly summarize some these aspects.

2. BEER basics

The model underlying the BEER metric is flexible for the integration of an arbitrary number of new features and has a training method that is targeted for producing good rankings among systems. Two other characteristic properties of BEER are its hierarchical reordering component and char n-grams lexical matching component.

BEER is essentially a linear model with which the score can be computed in the following way:

$$\text{score}(h, r) = \sum_i w_i \times \phi_i(h, r) = \vec{w} \cdot \vec{\phi} \quad (1)$$

where \vec{w} is a weight vector and $\vec{\phi}$ is a feature vector.

2.1. Learning-to-rank

Since the task on which our model is going to be evaluated is ranking translations it comes natural to train the model using *learning-to-rank* techniques.

Our training data consists of pairs of “good” and “bad” translations. By using a feature vector $\vec{\phi}_{\text{good}}$ for a good translation and a feature vector $\vec{\phi}_{\text{bad}}$ for a bad translation then using the following equations we can transform the ranking problem into a binary classification problem (Herbrich et al., 1999):

$$\begin{aligned}
 \text{score}(h_{\text{good}}, r) > \text{score}(h_{\text{bad}}, r) &\Leftrightarrow \\
 \vec{w} \cdot \vec{\phi}_{\text{good}} > \vec{w} \cdot \vec{\phi}_{\text{bad}} &\Leftrightarrow \\
 \vec{w} \cdot \vec{\phi}_{\text{good}} - \vec{w} \cdot \vec{\phi}_{\text{bad}} > 0 &\Leftrightarrow \quad (2) \\
 \vec{w} \cdot (\vec{\phi}_{\text{good}} - \vec{\phi}_{\text{bad}}) > 0 & \\
 \vec{w} \cdot (\vec{\phi}_{\text{bad}} - \vec{\phi}_{\text{good}}) < 0 &
 \end{aligned}$$

If we look at $\vec{\phi}_{\text{good}} - \vec{\phi}_{\text{bad}}$ as a positive training instance and at $\vec{\phi}_{\text{bad}} - \vec{\phi}_{\text{good}}$ as a negative training instance, we can train any linear classifier to find weight the vector \vec{w} that minimizes mistakes in ranking on the training set.

In practice BEER uses logistic regression as implemented in Weka toolkit (Hall et al., 2009). So the estimated weights are used in the following way:

$$\text{score}(h, r) = \frac{2}{1 + e^{-\sum_i w_i \times (\phi_i(h, r) - \phi_i(r, r))}} \quad (3)$$

The main difference from Equation1 is that here:

1. first subtract features of system translation given reference and reference given reference
2. apply sigmoid function and then
3. we multiply with 2

This formula does not make a difference in ranking compared to Equation 1 but it makes a difference in scaling the scores. Motivation for it is explained in Stanojević and Sima'an (2015). This scaling is important for getting a better corpus level score that is calculated as average sentence level score over whole corpus:

$$\text{BEER}_{\text{corpus}}(c) = \frac{\sum_{s_i \in c} \text{BEER}_{\text{sent}}(s_i)}{|c|} \quad (4)$$

2.2. Lexical component based on char n-grams

Lexical scoring of BEER relies heavily on character n-grams. Precision, Recall and F1-score are used with char n-gram orders from 1 until 6. These scores are more smooth on the sentence level than word n-gram matching that is present in other metrics like BLEU (Papineni et al., 2002) or METEOR (Michael Denkowski and Alon Lavie, 2014).

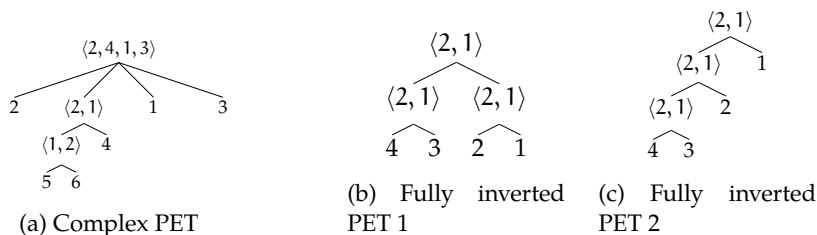


Figure 1: Examples of PETs

BEER also uses precision, recall and F1-score on word level (but not with word n-grams). Matching of words is computed over METEOR alignments that use WordNet, paraphrasing and stemming to have more accurate alignment.

We also make distinction between function and content words. For more precise description on used features and their effectiveness you can look at Stanojević and Sima'an (2014a).

2.3. Reordering component based on PETs

Alignment between system and reference translation can be simplified and considered as permutation of words from the reference translation in the system translation. Previous work by Isozaki et al. (2010) and Birch and Osborne (2010) used this permutation view of word order and applied Kendall τ for evaluating its distance from ideal (monotone) word order.

BEER goes beyond this *skip-gram* based evaluation and decomposes permutation into a hierarchical structure which shows how subparts of permutation form small groups that can be reordered all together. Figure 1a shows PET for permutation $\langle 2, 5, 6, 4, 1, 3 \rangle$. Ideally the permutation tree will be filled with nodes $\langle 1, 2 \rangle$ which would say that there is no need to do any reordering (everything is in the right place). BEER has features that compute the number of different node types and for each different type it assigns a different weight. Sometimes there are more than one PET for the same permutation. Consider Figure 1b and 1c which are just 2 out of 3 possible PETs for permutation $\langle 4, 3, 2, 1 \rangle$. Counting the number of trees that could be built is also a good indicator of the permutation quality.

3. Installing BEER

BEER is implemented in Scala so the only requirement for running it is just having the latest version of Java virtual machine installed (at least version 8). All the dependencies of BEER are included with the installation except METEOR and Stanford

CORE dependency parser (Chen and Manning, 2014) which gets installed automatically the first time BEER is ran.

The basic procedure to install BEER is with the following commands in terminal:

```
wget https://staff.fnwi.uva.nl/m.stanojevic/beer/beer_1.1.tar.gz
tar xfvz beer_1.1.tar.gz
./beer_1.1/beer # this installs METEOR and Stanford parser
rm beer_1.1.tar.gz
```

4. Usage from command line

BEER has several *working modes*. They specify if we want to use BEER for evaluation, for computing features, for training, for evaluating reordering or we want to use it for interactive evaluation on the terminal. We will explain three modes that are most useful from these: evaluation, evaluateReordering and interactive mode.

Bellow is an example of using BEER with evaluation working mode, where system translation is given with parameter `-s`, reference translation with parameter `-r` and language with parameter `-l`.

```
./beer --workingMode evaluation -l en -s system.en -r reference.en
```

Because `--workingMode evaluation` is the default setting we can also skip that parameter and just write:

```
./beer -l en -s system.en -r reference.en
```

This command will print the corpus level BEER score. To get the sentence level scores we just need to add `--printSentScores` to the command. The language parameter `-l` is an obligatory parameter for BEER because BEER uses language specific models for scoring and language specific resources (parsers, function words lists, paraphrase tables, stemmers...) for aligning reference and system translation. All languages from WMT13 and WMT14 are supported at this point. There is additional language *other* which is recommended in case there is no language specific model available.

This and some other parameters of BEER are shown in Table 1.

5. Usage in interactive mode

In some use cases the user might want to connect some other application with BEER. This can be done by using BEER as a library in case the other application executes on Java virtual machine, but the more general solution is usage of BEER through the interactive command line. This allows usage of BEER from any application which can read and write through pipe to some program. Here we describe this interactive way of using BEER.

| parameter | useage |
|----------------------|--|
| -l | input language |
| -s | system translation |
| -r | reference translations separated by column |
| --printSentScores | prints BEER score for each sentence |
| --printFeatureValues | prints feature values for each sentence |
| --norm | tokenizes the input using METEOR tokenizer |
| --noLower | stops BEER from lowercasing the input |
| --noPunct | excludes punctuation from evaluation |
| --help | prints these and some other parameters of BEER |

Table 1: Command line parameters of BEER

To start the interactive shell we need to set the working mode and the language for evaluation:

```
./beer --workingMode interactive -l other
```

When the interactive shell starts, we can type different commands for evaluation. To evaluate for a sentence level score we can type the following:

```
EVAL ||| system translation ||| reference 1 ||| reference 2
```

and then as output we should get the score for each reference translation. If we want only the best score out of all references we just need to type `EVAL BEST` instead of `EVAL`. In case we need feature values, the format is the same but we use `FACTORS` command instead of `EVAL`. Finally, to exit it is enough to type `EXIT`.

6. Statistical testing of BEER scores using MultEval

Usually it is not enough to know the final score of the system and whether it is better (or worse) than the baseline but also to know whether this difference is statistically significant. The tool that became quite popular for this task in MT community is Multeval (Clark et al., 2011) that has support for BLEU, TER and METEOR. With BEER we distribute the version of Multeval that contains a metrics module for BEER with the same interface as for the other metrics.

Here we describe how to use BEER with Multeval. Current implementation has no bugs that we are aware of, but it is relatively slow (it requires running in parallel and relatively large amount of RAM memory). This is likely to change soon.

Here is the command for running Multeval with BEER:

```
./multeval eval --metrics beer \
  --beer.language en \
  --refs references.en \
  --hyps-baseline translations.en.*
```

Basically the only additional obligatory parameter is `--beer.language`, but other than that all other parameters are standard Multeval parameters.

7. Tuning Moses for BEER – beta

BEER has support for tuning Moses (Koehn et al., 2007) systems parameters for higher BEER score. In principle all Moses optimization algorithms could be used, but we tested it only with *kbmira* (Cherry and Foster, 2012). For now, in order to add support for tuning Moses with BEER the user needs to recompile Moses by first adding files located in *src_moses* of BEER installation into Moses directory and then compiling. In future releases we hope to add support for BEER in the standard Moses installation so this manual compilation would not be necessary.

When Moses is compiled with the necessary C++ files, tuning can be done in the same way as usual by calling *mert-moses.pl* and by specifying BEER parameters in `--batch-mira-args` in the following way:

```
perl $MOSES/scripts/training/mert-moses.pl \
  --batch-mira-args="--sctype BEER --sconfig beerDir:$BEER_DIR,
  beerLang:en,beerModelType:tuning,beerThreads:30" ...
```

With *beerDir* we specify where Moses can find the installation of BEER that it can use. Standard models that are trained for human correlation have heavy recall bias and they are not good for MT tuning. That is why BEER has models without this bias that perform much better for tuning (Stanojević and Sima'an, 2015). To specify that we want to use that kind of model we use parameter *beerModelType* (currently only English is supported) and *beerLang* tells which language is evaluated.

8. Evaluating word order with PETs

BEER is a full evaluation metric that scores translation by all aspects (both fluency and adequacy). But sometimes we want to put more attention on the evaluation of either adequacy or fluency. Evaluating adequacy can be pretty straightforward, which is not the case for evaluating fluency. Metrics that are more successful in fluency treat this problem as measuring distance between the permutation of words in system translation from the ideal permutation (Birch and Osborne, 2010; Isozaki et al., 2010). In Stanojević and Sima'an (2014b) evaluation over permutations was extended from

| function | origin |
|--------------------|---|
| Kendall | Birch and Osborne (2010); Isozaki et al. (2010) |
| Spearman | Isozaki et al. (2010) |
| Ulam | Birch et al. (2010) |
| Hamming | Birch and Osborne (2010) |
| Fuzzy | Talbot et al. (2011) |
| PERecursiveViterbi | Stanojević and Sima'an (2014b) |
| PEFrecursive | Stanojević and Sima'an (2014b) |

Table 2: Implemented $\text{ordering}(\cdot, \cdot)$ functions

treating permutations as flat sequential structures to the hierarchical structures that better explain the reordering patterns.

In BEER installation, apart from standard trained BEER linear models, there is additionally an implementation of the following interpolation of lexical and ordering score:

$$\text{score}(s, r) = \alpha F_1(s, r) + (1 - \alpha) \text{ordering}(s, r) \quad (5)$$

F_1 is the lexical measure over unigrams and $\text{ordering}(\cdot, \cdot)$ is an ordering function over alignments (permutation) between words from system and reference translation. There are many implemented $\text{ordering}(\cdot, \cdot)$ functions shown in Table 2.

We can do the scoring with the following command:

```
./beer --workingMode evaluateReordering \  
      --alpha 0.5 \  
      --reorderingMetric PEFrecursive \  
      -l en -s system.en -r reference.en
```

Here α parameter is specified with `--alpha` and $\text{ordering}(\cdot, \cdot)$ function with `--reorderingMetric`. Other BEER parameters are mostly the same.

9. Summary

We have presented different ways in which BEER software can be used for evaluation and optimization of MT systems. We hope that this software package would increase usage of tunable metrics with state-of-the-art correlation with human judgment over standard metrics that are based on heuristics and usually perform badly on corpus and especially sentence level. BEER is licensed under GPL license and is available at <https://github.com/stanojevic/beer>.

Acknowledgements

This work is supported by STW grant nr. 12271 and NWO VICI grant nr. 277-89-002.

Bibliography

- Birch, Alexandra and Miles Osborne. LRscore for Evaluating Lexical and Reordering Quality in MT. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and Metrics-MATR*, pages 327–332, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W10-1749>.
- Birch, A., M. Osborne, and P. Blunsom. Metrics for MT evaluation: evaluating reordering. *Machine Translation*, pages 1–12, 2010. ISSN 0922-6567.
- Chen, Danqi and Christopher D Manning. A Fast and Accurate Dependency Parser using Neural Networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- Cherry, Colin and George Foster. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT '12*, pages 427–436, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.
- Clark, Jonathan H., Chris Dyer, Alon Lavie, and Noah A. Smith. Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2, HLT '11*, pages 176–181, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-88-6. URL <http://dl.acm.org/citation.cfm?id=2002736.2002774>.
- Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL <http://doi.acm.org/10.1145/1656274.1656278>.
- Herbrich, Ralf, Thore Graepel, and Klaus Obermayer. Support Vector Learning for Ordinal Regression. In *International Conference on Artificial Neural Networks*, pages 97–102, 1999.
- Isozaki, Hideki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. Automatic Evaluation of Translation Quality for Distant Language Pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP '10*, pages 944–952, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1870658.1870750>.
- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, ACL '07*, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
- Macháček, Matouš and Ondřej Bojar. Results of the WMT14 Metrics Shared Task. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 293–301, Baltimore, Maryland,

- USA, June 2014. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W/W14/W14-3336>.
- Michael Denkowski and Alon Lavie. Meteor Universal: Language Specific Translation Evaluation for Any Target Language. In *Proceedings of the ACL 2014 Workshop on Statistical Machine Translation*, 2014.
- Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02*, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL <http://dx.doi.org/10.3115/1073083.1073135>.
- Stanojević, Miloš and Khalil Sima'an. Fitting Sentence Level Translation Evaluation with Many Dense Features. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 202–206, Doha, Qatar, October 2014a. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/D14-1025>.
- Stanojević, Miloš and Khalil Sima'an. Evaluating Word Order Recursively over Permutation-Forests. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 138–147, Doha, Qatar, October 2014b. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W14-4017>.
- Stanojević, Miloš and Khalil Sima'an. BEER 1.1: ILLC UvA submission to metrics and tuning task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, Lisbon, Portugal, June 2015. Association for Computational Linguistics.
- Stanojević, Miloš, Amir Kamran, and Ondřej Bojar. Results of the WMT15 Metrics Shared Task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, Lisbon, Portugal, June 2015a. Association for Computational Linguistics.
- Stanojević, Miloš, Amir Kamran, and Ondřej Bojar. Results of the WMT15 Tuning Shared Task. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, Lisbon, Portugal, June 2015b. Association for Computational Linguistics.
- Talbot, David, Hideto Kazawa, Hiroshi Ichikawa, Jason Katz-Brown, Masakazu Seno, and Franz J. Och. A Lightweight Evaluation Framework for Machine Translation Reordering. In *Proceedings of the Sixth Workshop on Statistical Machine Translation, WMT '11*, pages 12–21, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-937284-12-1. URL <http://dl.acm.org/citation.cfm?id=2132960.2132963>.

Address for correspondence:

Miloš Stanojević

m.stanojevic@uva.nl

P.O. Box 94242, Amsterdam, The Netherlands



The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015 27-38

Box: Natural Language Processing Research Using Amazon Web Services

Amittai Axelrod

www.BoxResearch.ch

Abstract

We present a publicly-available state-of-the-art research and development platform for Machine Translation and Natural Language Processing that runs on the Amazon Elastic Compute Cloud. This provides a standardized research environment for all users, and enables perfect reproducibility and compatibility. Box also enables users to use their hardware budget to avoid the management and logistical overhead of maintaining a research lab, yet still participate in global research community with the same state-of-the-art tools.

1. Introduction

Amazon Web Services (AWS) is the umbrella term for all of the remote services that make up the Amazon cloud-computing platform. AWS includes the Elastic Compute Cloud (EC2) virtual computer cluster, as well as related services for storage, monitoring, analytics, and others.

We present our mechanism for using the Elastic Compute Cloud as the basis for a research and development platform for natural language processing researchers, students, and professionals. Using a cloud-based platform has numerous advantages. Compute nodes (“instances” in EC2 parlance) can start with the exact same software configuration, including installed tools, directories, and users. In particular, instances can be launched as direct clones of an existing virtual machine, or Amazon Machine Image (AMI). Our Box project creates such a disk image, containing open-source tools of use to the MT and NLP research community. As the disk images are frozen, identically-configured machines can be launched repeatedly, enabling stable and reproducible results. Future Box releases will be as separate AMIs, ensuring that

previous versions are always available, and previous experiments and code can always be re-run. Users have `sudo` access to instances they launch, so any Box machine can be updated or modified as desired.

While each instance has the same software configuration, each one can have a different hardware configuration, depending on the users' need. Box instances presently range from 1 vCPU with 3.75 Gb RAM (m3.medium) to 32 vCPU with 60Gb RAM (c3.8xlarge). The next major release of Box will expand to support a larger range, from 1 vCPU with 1Gb RAM through 32-40 vCPU with 160-244Gb RAM.

The downside is that while the compute nodes and the installed tools are open-source and free ("libre, as in software"), running the EC2 instances themselves is not free ("gratis, as in beer"). Nonetheless, cloud-based computing now enables researchers to spend their hardware budget on an as-needed basis, and scale their experimentation accordingly, rather than paying upfront to purchase, set up, and maintain a fixed physical cluster. As such, working on Box (or EC2 in general) can yield higher experimental throughput for a fixed budget.

Box itself comes with many SMT tools pre-installed, with the goal of enabling immediate productivity and lowering the barrier to entry. Furthermore, rather than setting up the complete pipeline for one MT toolkit, as do most research labs, Box comes with both the Moses (Koehn et al., 2007) and cdec (Dyer et al., 2010) pipelines, as well as compatible tools: `fast_align` (Dyer et al., 2013), `mgiza` (Gao and Vogel, 2008), `kenlm` (Heafield, 2011), and Jonathan Clark's ducttape. We will shortly include Joshua (Li et al., 2009), as well as other MT tool suites. By doing so, we hope to enable head-to-head comparisons of MT tools as well as mixing-and-matching. This provides a reason to avoid parallel development, and reducing duplicated effort.

Box also includes other non-core MT tools to stimulate exploration. This release includes `multeval` (Clark et al., 2011), `rnnlm` (Mikolov et al., 2010), and `word2vec` (Mikolov et al., 2013), with the intent to expand the list shortly. There are many open-source tools that each perform one NLP-related task well, such as speech recognition, optical character recognition, text-to-speech, dialog systems, and so on. However, there are few end-to-end open-source multilingual systems available for desirable tasks such as translating visual input from a smartphone camera, spoken announcements in subway stations, or dialog in real-time. We hope that by providing easy access to available components, we can accelerate progress.

2. Case Study

Box was intended from the beginning to be used by researchers without sufficient computational resources to work effectively. Lacking access to a cluster, the experimental results in our dissertation (Axelrod, 2014) were originally scoped for a single 4-core desktop machine. However, that allowed training only one or two MT systems per week; not conducive to training a thousand systems and graduating efficiently. We therefore architected a workflow to use Amazon's Elastic Compute Cloud

to mimic a traditional filesystem and cluster. For financial reasons, we also prioritized not leaving machines idle.¹ This workflow is independent of Box itself: each Box instance is also designed to be launched and used as a standalone development machine. What follows is only one possible way of using Box.

2.1. Instance Specifications

We primarily used three sizes of general-purpose Amazon EC2 instances. Specs for the current generation of instances are listed in Table 1.² It is worth mentioning that instance pricing appears to decrease slightly over time. A medium instance suffices for training a Moses system on TED-sized data (150k sentence pairs) in a few hours, for a negligible cost. A medium instance can also be used to run just the Moses decoder using translation models that have been filtered down to match a particular test set. A large instance is suitable for training up to around 1.5M sentence pairs, or most of Europarl, in under a day. For larger corpora we used the xlarge instance type, and found that the extra cores meant that 2M sentences trained faster on an xlarge than 1M on a large.

| Name | vCPU | Memory (GB) | Hourly Cost |
|-----------|------|-------------|-------------|
| m3.medium | 1 | 3.75 | \$0.067 |
| m3.large | 2 | 7.5 | \$0.133 |
| m3.xlarge | 4 | 15 | \$0.266 |

Table 1. Basic Amazon EC2 instance specifications.

2.2. Workflow

The Amazon Elastic Compute Cloud can be used effectively while only running the smallest (and cheapest) possible machine (a *.micro instance) full-time. The cluster nodes – termed EC2 instances – cost more, and thus are launched for particular experiments and terminated immediately afterward. The micro instance (the *home instance*) is a regular Linux machine, not a Box clone, and it acts as a gateway machine. The home instance is the white box in the upper right of Figure 1.

The home instance has a very large storage volume attached (volumes are only accessible if they are mounted to a running instance). This large volume, shown as a large circle in the lower right of Figure 1, acts as central filesystem storage for corpora

¹ If money is no object, then one could launch 20 EC2 instances, and run them year-round without worrying about idle nodes. However, it is likely cheaper to buy the hardware and hire a sysadmin.

²A Box instance costs 10% more to use per hour than a bare-bones instance, but all other Amazon fees for data transfer and storage are unchanged.

and experimental results. This avoids the need to upload or download large amounts of data to the Box instances. Amazon charges users to transfer data, and transferring data between EC2 instances is faster and cheaper than between EC2 and the non-Amazon world. As such, it is more efficient to transfer all data at once.

Corpora that are going to be passed to multiple Box instances should be stored in a data snapshot (shown as the grey circle in the lower left of Figure 1). This snapshot can then be cloned into new volumes for each experimental node. These cloned volumes are the white circles in the middle of Figure 1.

Amazon Machine Images (the AMI is the grey box in the upper left of Figure 1) are snapshots of computers. The AMI is used to launch new machines called *instances*, which are the white boxes in the middle of Figure 1. While the instances can have different hardware configurations, they all have the same installed software and setup.

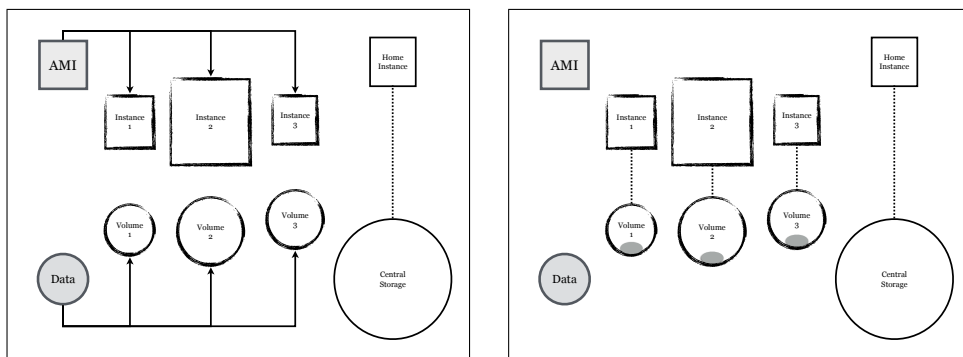


Figure 1. (Left) Launching instances from an AMI and cloning storage volumes from a data snapshot. (Right) After the jobs finish, experimental output is stored on the volume attached to each node.

The new volumes are attached to the new instances, and the experiments are run. Box instances are configured to be ordinary (yet powerful) computers, so experimentation can either be done by manually logging in to each instance and issuing commands in a normal session, or via writing the entire experiment as a single shell script and then executing the job on the Box instance. After the jobs finish, we store the experimental output on the volume attached to each node. The results are illustrated as grey content on the white data volumes in Figure 1. Having each instance work in a directory on an attached volume protects the results from accidental termination of the instance, which deletes all data stored on the instance itself. It is also a cost-saving

measure, as some cheap instances have little disk space included, and an additional volume is much cheaper than a larger instance.³

The experimental results are copied from the attached volumes to the central storage volume, and then the instances are killed and the volumes deleted, as shown in Figure 2.

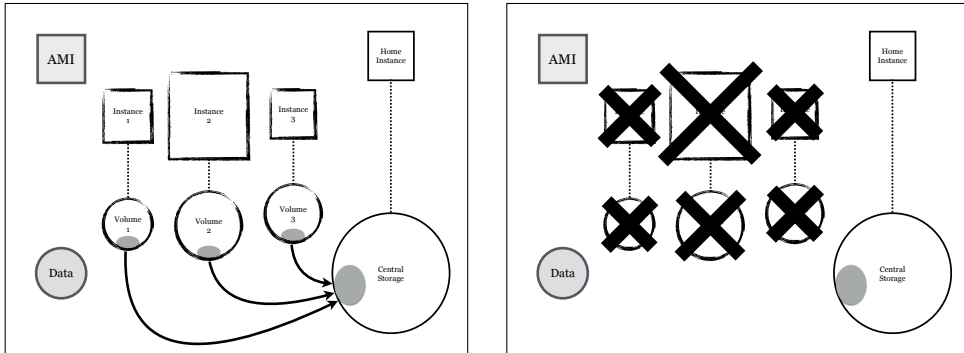


Figure 2. (Left) Experimental output is copied from each instance to central storage. (Right) All instances and attached volumes are deleted.

2.3. Reproducibility

We used the first (alpha) version of Box for all of the dissertation experiments, run over a two year span. While revising the final document, we were able to re-run experiments from 18 months prior and compare against newer work. This reproducibility is difficult to achieve even when a researcher has complete control of their development environment, as once tools are updated they are rarely rolled back even if they could be. With Box this is easy, as the development environment can be updated as needed, but older versions remain frozen yet in running order.

3. Signing up for the Elastic Compute Cloud

To work on the Amazon Web Services cloud, users need: an Amazon.com account, a telephone number (for account confirmation), and a method of payment and billing address. There is no charge to set up an account, and there is a year-long free usage tier

³ One gigabyte-month of storage costs less than one compute-hour.

for new customers. There are how-to videos⁴ for creating an AWS account, launching instances, and more.

3.1. Setup process

The setup process⁵ must be done only once, and is as follows:

1. Make an AWS account.

Visit `aws.amazon.com`, click “sign up”,⁶ and fill out the forms as instructed.

2. Create AWS security keys (optional).

These are for using the command-line interface to the AWS account instead of the browser. They are not necessary if the user prefers to use only the browser’s GUI console to launch and manage their cluster usage. The names “Access Key” and “Secret Access Key” as used by Amazon may be confusing. They are simply a username/password pair that can be changed independently of the AWS account name and password. The keys can be generated from the Identity and Access Management (IAM) console, via User Actions → Manage Access Keys → Create Access Key⁷.

3. Get SSH keys.

This is a standard RSA keypair, necessary to log into EC2 instances. It is created via the EC2 console (`https://console.aws.amazon.com/ec2/`), under Navigation → Network & Security → Key Pairs.⁸ The private key will automatically download – only once! – as a *.pem file. Failing to save the private key will render the ssh keypair useless and the process must be repeated.⁹

The private key must have specific file permissions, and set as follows:

```
chmod 400 $PRIVATE_SSH_KEYFILE
```

3.2. A Note on EC2 and Geography

Amazon’s Elastic Compute Cloud is hosted in several locations around the globe, forming independent sets of resources. *Regions* are top-level geographical distinctions

⁴ Video guides: <http://aws.amazon.com/getting-started/>

⁵ Setup guide: <http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-set-up.html>

⁶ Or go directly to <https://portal.aws.amazon.com/gp/aws/developer/registration/index.html>

⁷ IAM Console: <https://console.aws.amazon.com/iam/home?#home>

See <http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-set-up.html#cli-signup>

⁸Creating SSH keys:

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html#having-ec2-create-your-key-pair>

⁹Note that the Amazon documentation refers to the SSH private key file as `my-key-pair.pem`. This implies the existence of an keypair called `my-key-pair`, but the `my-key-pair.pem` file contains *only* the private key and not the public one! This is not necessarily what might be expected. To avoid confusion, we refer to the keypair as `SSH_KEYPAIR` and the private key file as `PRIVATE_SSH_KEYFILE`, with the understanding that `PRIVATE_SSH_KEYFILE = "SSH_KEYPAIR.pem"`.

(e.g. “U.S. East”, “South America”). By default, nothing is replicated across multiple regions, but any user can access any region regardless of where they are located, so the current version of Box is available only in the us-east-1 region. The primary differences between EC2 regions are latency, and resource cost¹⁰.

Availability Zones are subdivisions within a single EC2 Region, and named accordingly (e.g. us-east-1a, us-east-1b). By default Amazon will load-balance the availability zones, resulting in some instances launching in one availability zone, and some in another. It can be useful to have everything within a single availability zone to ensure complete interoperability, particularly if volumes will be each attached to and detached from multiple instances. The examples in this work assume the user is using the AWS us-east-1 region and the us-east-1a availability zone, regardless of where in the world the user is physically located. The us-east-1 region is required for the current release of Box, but can be changed to adapt the instructions to other AMIs. The choice of availability zone here is arbitrary, and can be set at whim.

4. Signing up for Box

Accessing a community-made *paid* AMI such as Box requires first adding the associated *product* to the user’s account.¹¹ This can be used to grant access to a number of related AMIs at once. At present there is only one Box AMI (ami-1d678876, v2015.05.26) in the product, but each future release will be a separate AMI. This ensures perfect reproducibility, as new Box AMIs provide expanded and updated research tools, but previous Box releases remain static and accessible. An experiment done once on a fresh Box instance can always be done on an instance of that Box.¹²

Amazon does the association of AWS account to Box product via a purchase for \$0.00. As such, users are asked to confirm the payment method and shipping address from their AWS signup, though nothing is charged and nothing will be shipped. The link to sign up for Box is here: <https://portal.aws.amazon.com/gp/aws/user/subscription/index.html?offeringCode=49B2A70F>. Signing up for Box is free (as in beer). Only the actual use of Box incurs charges. If accessing other virtual machines on EC2, the cost-conscious researcher should note that other AWS EC2 products may have upfront fees or monthly charges in addition to (or instead of) usage costs.

¹⁰ The us-east-1 region has historically been the cheapest.

¹¹ Ironically, the Amazon server that handles these transactions is slow, and each page takes 30 seconds to load. Fortunately this process needs to be done only once.

¹² As long as the underlying Linux distribution of the Box (or any) AMI remains compatible with the hardware used by EC2. At present this is not an issue. All releases of the Amazon Linux AMI (since it exited Beta in 2011) are still runnable, even the 32-bit ones, indicating Amazon intends to preserve compatibility.

5. Basic AWS EC2 Operations

Few commands are necessary to use EC2 as a research resource, whether with Box or some other virtual machine. It is useful to know how to:

1. Launch a new instance from an existing Amazon Machine Image (AMI)
2. Create a new disk volume (optionally from a snapshot of an existing volume)
3. Attach a volume to an instance
4. Delete a volume
5. Terminate an instance

Of these, only launching and terminating instances are requirements. Being able to create, delete, and attach new volumes¹³ is helpful for cloning disk volumes that already contain corpora, so as to avoid having to copy data over every time.

5.1. AWS EC2 Web Console

All of the instance management for EC2 can be done from the AWS EC2 console in a web browser¹⁴ (Chrome, Firefox, Internet Explorer, and Safari are supported).

5.1.1. Launching a new Box instance via the console

The easiest method of launching a new instance is to use right-click on a running instance and select “Launch another instance like this one”. The second-easiest method is to bookmark the URL of the launch wizard for a new Box instance:

<https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:ami=ami-1d678876>

The wizard presents the user with a series of steps:

Instance Type: For building SMT systems, it is important to pick an instance with enough memory for the size of the training corpus. Many configuration options can be found by scrolling down; greyed-out machine types are not available.¹⁵

Configure Instance The VPC (Virtual Private Cloud) *network type* is the default option and easiest for new users. Any *availability zone* will work, but creating all instances and volumes in the same zone is best (see Section 3.2). *Monitoring* is not useful.¹⁶

¹³ Volumes detach automatically when the instance they are mounted on is terminated.

¹⁴ EC2 Console: <https://console.aws.amazon.com/ec2/v2/home>

¹⁵ Complete list of EC2 instance types: <https://aws.amazon.com/ec2/instance-types/>
 Complete list of EC2 instance prices: <https://aws.amazon.com/ec2/pricing/>

¹⁶Monitoring tracks instance state, such as uptime of a customer-facing webservice, but increases costs.

Additional Storage Some instance types do not include much disk space. Disk is cheap, so it is easy to add any desired amount. However, all of this storage is deleted when the instance is terminated. A useful alternative is to not add additional storage to the instance directly, and instead create a new EC2 disk volume with plenty of space and attach it to the instance (see Sections 5.1.2 and 5.1.3). This prevents accidentally losing data or experimental results when the instance is terminated.

Tag Instance It can be difficult to differentiate instances (and attach a volume to the correct one, for example) when many are running. Unlike nodes in a regular cluster, each instance can be given a tag to indicate the experiment or process that is running on it. Prepending the date to the instance labels (e.g. “150524 building Box beta”) can help keep the list of instances sorted.

Security Group Box instances are a good way to work on restricted or sensitive datasets, as corpora can be kept completely separate from other clients’ or users’ data. That being said, while crucial in fixed-location or long-duration applications, security settings are less important for a machine that will be terminated sooner rather than later. The default of “all ports, all sources” is good enough for casual users logging in with `ssh -i $PRIVATE_SSH_KEY.pem`. Diligent users can restrict access to port 22 (SSH) only, or permit only a particular port from a particular IP address.

After Launch The final step is to launch the instance and log in. See Section 5.3.

5.1.2. Creating a new storage volume via the console

The AWS console’s sidebar has a section titled “Elastic Block Store”. Clicking “Volume” → “Create Volume” will pop up a small wizard for making a new volume that will exist independently of any instances. For *Type*, “General Purpose SSD” will suffice. Select the size as needed, and the Availability Zone to match other instances and volumes already created. To clone a volume that already contains the desired corpora or models, enter the *Snapshot ID*. If the volume created is larger than the snapshot it is created from, then the volume will need to be resized with `sudo resize2fs` after it is attached to a running instance.

5.1.3. Attaching a volume to a running Box instance via the console

Independent storage volumes must be attached to – and then mounted on – running instances in order to be accessed. In the Volumes view of the EC2 Console, right-clicking any Volume ID produces a menu with the option to *Attach Volume*. Enter the target instance (the instance name tags from Section 5.1.1 are useful here). Note that the instance must be in the same Availability Zone. The last option is to select

a Linux device name for the volume. Single-use instances can generally be expected to have only one attached volume, so attaching all volumes to the same location (e.g. `/dev/xvdf`) allows all instances to run the same setup scripts. After attaching the volume, it must be mounted. This is done with `sudo mount` after logging in to the instance.

5.1.4. Deleting a volume

The same right-click menu on a Volume ID contains an option to *Delete Volume*. This should only be done after copying all experimental output off of the volume.

5.1.5. Terminating a Box instance via the console

The EC2 Console includes an Instances view, selectable via the sidebar. Right-clicking any Instance ID shows a menu which includes an option to *Terminate Instance*. This kills the instance immediately, deletes everything stored on the instance. Any attached volumes are unmounted and detached, and their contents are preserved.¹⁷

5.2. The Command Line Interface

Experienced researchers may appreciate being able to script these interactions via the Amazon Web Services Command Line Interface (AWS CLI) tool. The AWS CLI tool is itself released¹⁸ under the Apache 2.0 licence, making it compatible with both derivative commercial works and the open source community. A detailed guide to using the AWS CLI for managing instances can be found on our website.

5.3. Logging in to a Box Instance

All EC2 instances take 2-5 minutes to boot after launch. After the instance finishes booting, its status will change to green in the console. The next step in using Box is to access the instance. Either its public IP address or DNS name, found by clicking on the instance in the EC2 Instances view, can be used to log in. The default username is `ec2-user`, thus:

```
ssh -i $PRIVATE_SSH_KEYFILE ec2-user@$IP_ADDRESS
```

Once the user is logged in to the instance, there are some optional steps to finish the instance setup. The first is to add some swap space, as by default there is none:

```
sudo dd if=/dev/zero of=/media/ephemeral0/swapfile bs=3M count=1024
sudo mkswap /media/ephemeral0/swapfile
sudo swapon /media/ephemeral0/swapfile
sudo swapon -s
```

¹⁷This is the advantage of using attached volumes when working on EC2 instances.

¹⁸ AWS Git repo: <https://github.com/aws/aws-cli>

The second step is to mount any attached volumes:

```
mkdir -p ~/exp/  
sudo mount /dev/xvdf ~/exp/  
sudo resize2fs /dev/xvdf
```

Recall the suggestion that for ease of management, all volumes attached as in Section 5.1.3 be labeled `/dev/xvdf`. These commands can then be placed at the beginning of every experiment script and run automatically on all instances.

The Box instance is now ready for use.

6. Collaborating with Box

Multiple users can share a single Box instance. For a quick look around on an instance (*e.g.* to help debug a lab project on a student’s Box), the `$PRIVATE_SSH_KEYFILE` file can be shared with the instructor. As the default username is the same for each EC2 instance (`ec2-user`), this is a very fast and simple way of sharing access to an instance. However, this presumes a trusted relationship already exists between the two parties. Sharing keys is otherwise anti-recommended for semi-permanent instances open to the outside world, or where multiple users’ work may collide.

For longer or more formal collaborations, it is better to add new users. All Box instances are Linux machines with `sudo` privileges for the main user, so this can be done at will.¹⁹ Each new user has their own private ssh key, and can log in separately just like any other server. The advantage, of course, is that granting access to a shared research instance does not involve granting access to an entire filesystem within the users’ organizations. In this way it is possible for researchers to collaborate across institutional (or academic/industry) administrative boundaries, or to teach an online course to students who are not formally enrolled in the instructors’ department.

7. Conclusion

The current release of Box provides access to a state-of-the-art research and development environment to both experienced and new researchers alike. Box runs on the Amazon Elastic Compute Cloud, so it can be used to provide computational resources to those who have none, or to supplement an existing cluster. Standard open-source toolkits for machine translation and natural language processing are pre-installed on Box, putting any user in the position to start work immediately. This enables students, researchers, and developers to contribute to the field without being limited by their computational resources at hand. By this mechanism we hope to substantially lower the barrier to entry for the field of NLP.

¹⁹Adding users: <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/managing-users.html>

Acknowledgements

The author appreciates the early encouragement of Achim Ruopp, who made the first Moses installation for EC2.

Bibliography

- Axelrod, Amittai. *Data Selection for Statistical Machine Translation*. PhD thesis, University of Washington, 2014.
- Clark, Jonathan H, Chris Dyer, Alon Lavie, and Noah Smith. Better Hypothesis Testing for Statistical Machine Translation : Controlling for Optimizer Instability. *ACL (Association for Computational Linguistics)*, 2011.
- Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Ture, Phil Blumson, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. *ACL (Association for Computational Linguistics) Interactive Poster and Demonstration Sessions*, 2010.
- Dyer, Chris, Victor Chahuneau, and Noah A Smith. A Simple, Fast, and Effective Reparameterization of IBM Model 2. *NAACL (North American Association for Computational Linguistics)*, 2013.
- Gao, Qin and Stephan Vogel. Parallel Implementations of Word Alignment Tool. *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, 2008.
- Heafield, Kenneth. KenLM : Faster and Smaller Language Model Queries. *WMT (Workshop on Statistical Machine Translation)*, 2011.
- Koehn, Philipp, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Christine Moran, Chris Dyer, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. *ACL (Association for Computational Linguistics) Interactive Poster and Demonstration Sessions*, 2007.
- Li, Zhifei, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren N G Thornton, Jonathan Weese, and Omar F Zaidan. Joshua: An Open Source Toolkit for Parsing-based Machine Translation. *WMT (Workshop on Statistical Machine Translation)*, 2009.
- Mikolov, T, M Karafiat, L Burget, J Cernocky, and S Khudanpur. Recurrent Neural Network based Language Model. *INTERSPEECH*, 2010.
- Mikolov, Tomas, Quoc V Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. *arXiv preprint arXiv:1309.4168v1*, 2013.

Address for correspondence:

Amittai Axelrod
amittai.box@gmail.com
4423 Lehigh Rd #666,
College Park, MD 20740, USA



The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015 39-50

Sampling Phrase Tables for the Moses Statistical Machine Translation System

Ulrich Germann

University of Edinburgh

Abstract

The idea of virtual phrase tables for statistical machine translation (SMT) that construct phrase table entries on demand by sampling a fully indexed bitext was first proposed ten years ago by Callison-Burch et al. (2005). However, until recently (Germann, 2014) no working and practical implementation of this approach was available in the *Moses* SMT system.

We describe and evaluate this implementation in more detail. Sampling phrase tables are much faster to build and are competitive with conventional phrase tables in terms of translation quality and speed.

1. Introduction

Phrase-based statistical MT translates by concatenating phrase-level translations that are looked up in a dictionary called the *phrase table*. In this context, a phrase is any sequence of consecutive words, regardless of whether or not it is a phrase from a linguistic point of view. In addition to the translation options for each phrase, the table stores for each translation option a number of scores that are used by the translation engine (*decoder*) to rank translation hypotheses according to a statistical model.

In the *Moses* SMT system, the phrase table is traditionally pre-computed as shown in Fig. 1. First, all pairs of phrases up to an arbitrary length limit (usually between 5 and 7 words), and their corresponding translations are extracted from a word-aligned parallel corpus, using the word alignment links as a guide to establish translational correspondence between phrases. Phrase pairs are scored both in the forward and backward translation direction, i.e., $p(\textit{target}|\textit{source})$ and $p(\textit{source}|\textit{target})$, respectively. Computing these scores is traditionally done by sorting the lists on disk first to facili-

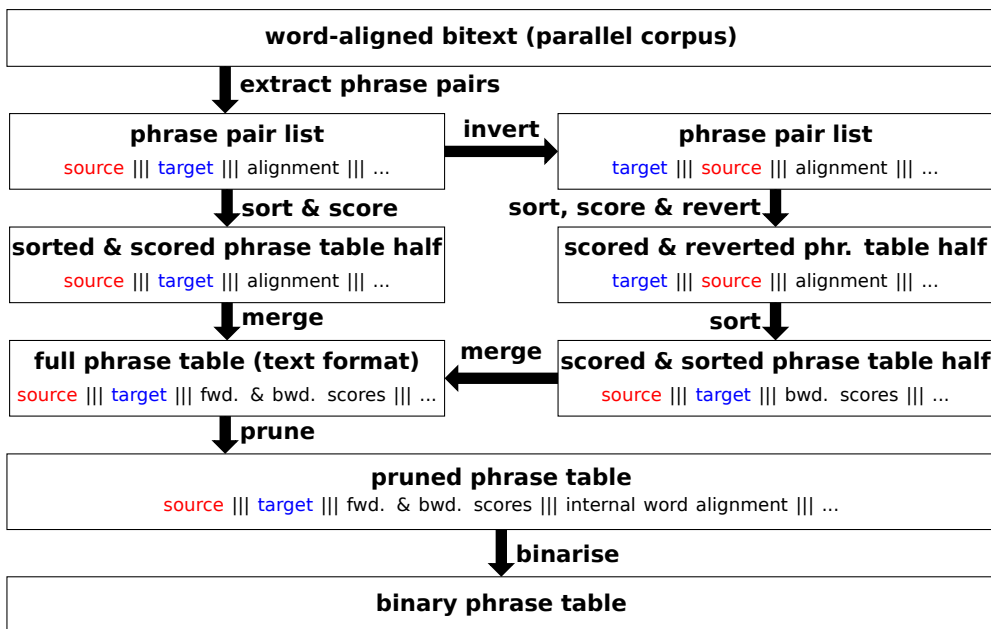


Figure 1. Conventional Phrase Table Construction

tate the accumulation of marginals. This approach requires sorting the list of extracted phrase pairs at least twice: once to obtain joint and marginal counts for estimation of the forward translation probabilities, and once to calculate the marginals for the backward probabilities. In practice, forward and backward scoring take place in parallel, as shown in Figure 1.

The resulting phrase tables often have considerable levels of noise, due to misaligned sentence pairs or alignment errors at the word level. Phrase table *pruning* removes entries of dubious quality. Even with pruning, conventional phrase tables built from large amounts of parallel data are often too large to be loaded and stored completely in memory. Therefore, various binary phrase table implementations were developed in *Moses* over the years, providing access to disk-based data base structures (Zens and Ney, 2007)¹ or using compressed representations that can be mapped into memory and “unpacked” on demand (Junczys-Dowmunt, 2012).

¹The original implementation by R. Zens (*PhraseDictionaryBinary*) has recently been replaced in *Moses* by *PhraseDictionaryOnDisk* (H. Huang, personal communication).

Due to the way they are built, conventional phrase tables for *Moses* are fundamentally static in nature: they cannot be updated easily without repeating the entire costly creation process.

2. Phrase tables with on-demand sampling

As an alternative to pre-computed phrase tables, Callison-Burch et al. (2005) suggested the use of suffix arrays (Manber and Myers, 1990) to index the parallel training data for full-text search, and to create phrase table entries on demand at translation time, by sampling in the bitext occurrences of each source phrase in question, extracting counts and statistics as necessary.

A suffix array over a corpus $\langle w_1, \dots, w_n \rangle$ is an array $\langle 1 \dots n \rangle$ of all token positions in that corpus, sorted in lexicographic order of the token sequences that start at the respective positions. Figure 2 shows a letter-based suffix array over the word ‘suffixarray’. For bitext indexing for MT, we index at the word level.

Given a suffix array and the underlying corpus, we can easily find all occurrences of a given search sequence by performing a binary search in the array to determine the first item that is greater or equal to the search sequence, and a second search to find the first item that is strictly greater. Every item in this sub-range of the array is the start position of an occurrence of the search sequence in the corpus. From this pool of occurrences, we extract phrase translations for a reasonably large sample using the usual phrase extraction heuristics.

Lopez (2007, 2008) explored this approach in detail in the context of *hierarchical phrase-based translation* (Chiang, 2007). Schwartz and Callison-Burch (2010) implemented Lopez’s methods in the *Joshua* decoder (Li et al., 2009). Suffix array-based translation rule extraction is also used in *cdec* (Dyer et al., 2010). However, until recently (Germann, 2014), no efficient, working implementation of sampling phrase tables was available in the *Moses* decoder. The purpose of this article is to document this implementation in detail, and to present results of empirical evaluations that demonstrate that sampling phrase tables are an attractive, efficient, and competitive alternative to conventional phrase tables for phrase-based SMT.

The apparent lack of interest in sampling phrase tables in the phrase-based SMT community may have been partly due to the fact that naïve implementations of the approach tend perform worse than conventional phrase tables. To illustrate this point, we repeat in Table 1 the results of a comparison of systems from Germann (2014). Several German-to-English systems were constructed with conventional and sampling phrase tables. All systems were trained on ca. 2 million parallel sentence pairs from Europarl (Version 7) and the News Commentary corpus (Version 9), both available

| | | | | |
|----|--|------------|--|-------------|
| 7 | | suffix | | array |
| 10 | | suffixarr | | ay |
| 3 | | su | | ffixarray |
| 4 | | suf | | fixarray |
| 5 | | suff | | ixarray |
| 9 | | suffixar | | ray |
| 8 | | suffixa | | rarray |
| 1 | | | | suffixarray |
| 2 | | s | | uffixarray |
| 6 | | suffi | | xarray |
| 11 | | suffixarra | | y |

Figure 2. Letter-based suffix array over the word ‘suffixarray’

| # | method | low | high | median | mean | 95% conf. interval ^a | runs |
|---|--|-------|-------|--------|-------|---------------------------------|------|
| 1 | precomp., Kneser-Ney smoothing | 18.36 | 18.50 | 18.45 | 18.43 | 17.93–18.95 | 10 |
| 2 | precomp., Good-Turing smoothing | 18.29 | 18.63 | 18.54 | 18.52 | 18.05–19.05 | 10 |
| 3 | precomp., Good-Turing smoothing, filtered^b | 18.43 | 18.61 | 18.53 | 18.53 | 18.04–19.08 | 10 |
| 4 | precomp., no smoothing | 17.86 | 18.12 | 18.07 | 18.05 | 17.58–18.61 | 10 |
| 5 | max. 1000 smpl., no sm., no bwd. prob. | 16.70 | 16.92 | 16.84 | 16.79 | 16.35–17.32 | 10 |
| 6 | max. 1000 smpl., no sm., with bwd. prob. | 17.61 | 17.72 | 17.69 | 17.68 | 17.14–18.22 | 8 |
| 7 | max. 1000 smpl., $\alpha = .05$, with bwd. prob.^c | 18.35 | 18.43 | 18.38 | 18.38 | 17.86–18.90 | 10 |
| 8 | max. 1000 smpl., $\alpha = .01$, with bwd. prob. | 18.43 | 18.65 | 18.53 | 18.52 | 18.03–19.12 | 10 |
| 9 | max. 100 smpl., $\alpha = .01$, with bwd. prob. | 18.40 | 18.55 | 18.46 | 18.46 | 17.94–19.00 | 10 |

table adapted from Germann (2014)

^a computed via bootstrap resampling for the median system in the group.

^b top 100 entries per source phrase selected according to $p(t|s)$.

^c α : one-sided confidence level of the Clopper-Pearson confidence interval for the observed counts.

Table 1. BLEU scores (*de* → *en*) with different phrase score computation methods.

from the web site of the 2014 *Workshop on Statistical Machine Translation (WMT)*² They were tuned on the NewsTest 2013 data set, and evaluated on the NewsTest 2014 data set from the Shared Translation Tasks at WMT-2013 and WMT-2014, respectively. The systems differ in the number of feature functions used (with and without backwards phrase-level translation probabilities) and smoothing methods applied. No lexicalized reordering model was used in these experiments.

Each system was tuned 8-10 times in independent tuning runs with Minimum Error Rate Training (MERT; Och, 2003). Table 1 shows low, high, median, and mean scores over the multiple tuning runs for each system.

The first risk in the use of sampling phrase tables is that it is tempting to forfeit the backwards phrase-level translation probabilities in the scoring. The basic sampling and phrase extraction procedure produces source-side marginal and joint counts over a sample of the data, but not the target-side marginal counts necessary to compute $p(\text{source}|\text{target})$. These backwards probabilities are, however, important indicators of phrase-level translation quality, and leaving them out hurts performance, as illustrated by a comparison of Lines 2 and 5 in Tab. 1 (standard setup vs. naïve implementation of the sampling approach without backward probabilities and smoothing).

While it is technically possible to “back-sample” phrase translation candidates by performing the sampling and gathering of counts inversely for each phrase translation candidate, this would greatly increase the computational effort required at translation time, and slow down the decoder. A convenient and effective short-cut, however, is to simply scale the target-side global phrase occurrence counts of each translation

²<http://www.statmt.org/wmt14/translation-task.html#download>

candidate by the proportion of sample size to total source phrase count:

$$p(\text{source} | \text{target}) \approx \frac{\text{joint phr. count in sample}}{\text{total target phr. count}} \cdot \frac{\text{total source phr. count}}{\# \text{ of source phr. sampled}} \quad (1)$$

As Line 6 in Tab. 1 shows, this method narrows the performance gap between conventional systems and sampling phrase tables, although it does not perform as well as “proper” computation of the backwards probabilities (cf. Line 4 in the table).

The second disadvantage of the sampling approach is that it cannot use the standard smoothing techniques used to compute smoothed phrase table scores in conventional phrase tables, i.e. Good-Turing or Kneser-Ney, as these require global information about the phrase table that is not available when sampling. The results in Lines 4 and 6 (vs. Line 2) confirm the finding by Foster et al. (2006) that phrase table smoothing improves translation quality.

One particular problem with maximum likelihood (ML) estimates in the context of translation modeling is the over-estimation of the observations in small samples. The smaller the sample, the bigger the estimation error. Since the decoder is free to choose the segmentation of the input into source phrases, it has an incentive to pick long, rare phrases. The smaller sample sizes result in bigger over-estimation of the true translation probabilities. This in turn leads to higher model scores, which is what the decoder aims for. Alas, in this case higher model scores usually do not mean higher translation quality — ML estimates introduce modelling errors. Smoothing dampens this effect.

In lieu of the established smoothing techniques, we counteract the lure of small sample sizes by replacing maximum likelihood estimates with the lower bound of the binomial confidence interval (Clopper and Pearson, 1934) for the observed counts in the actual sample, at an appropriate level of confidence.³ Figure 3 shows the “response curve” of this method for a constant success rate of $1/3$, as the underlying sample size increases. In practice, a confidence level of 99% appears to be a reasonable choice: in our German-to-English experiments, using the lower bound of the binomial confidence interval at this level brought the BLEU performance of the system with a sampling phrase table back to the level of decoding with a conventional phrase table (cf. Line 8 vs. Line 2 in Tab. 1).

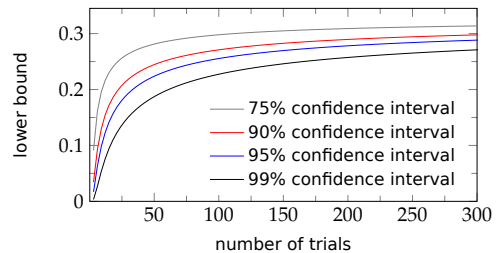


Figure 3. Lower bound of binomial confidence interval for success rate $\frac{1}{3}$

³An alternative is the use of additional features that keep track of quantized raw joint phrase counts, e.g., how many phrase translations used in a translation hypothesis were actually observed at most once, twice, three times, or more frequently (Mauser et al., 2007).

Another concern about sampling phrase tables is speed. After all, phrase extraction and assembly of phrase table entries on the fly do require more computation at translation time. However, caching of entries once created as well as multi-threaded sampling make the current implementation of sampling phrase tables in *Moses* very competitive with their alternatives.

A comparison of translation times for a large French–English system with sampling phrase tables vs. the compressed phrase tables of Junczys-Dowmunt (2012) (*CompactPT*) is given in Tab. 3 (Sec. 7) below. *CompactPT* is the fastest phrase table implementation available in *Moses* for translation in practice.

3. Lexicalised reordering model

Lexicalized reordering models improve the quality of translation (Galley and Manning, 2008). Sampled lexicalised reordering models were not available for the work presented in Germann (2014), but have been implemented since. Our sampling procedure keeps track of the necessary information for hierarchical lexicalized reordering (Galley and Manning, 2008) and communicates this information to the lexicalised reordering model.

4. Dynamic updates

One special feature of the sampling phrase table implementation in *Moses* is that it allows to add parallel text dynamically through an RPC call when *Moses* is run in server mode. This is useful, for example, when *Moses* serves as the MT back-end in an interactive post-editing scenario, where bilingual humans post-edit the MT output. Dynamic updates allow immediate exploitation of the newly created high-quality data to improve MT performance on the spot.

To accommodate these updates, the phrase table maintains two separate bitexts: the memory-mapped, static *background* bitext whose build process was just described, and a dynamic *foreground* bitext that is kept entirely in-memory. The phrase table’s built-in feature functions can be configured to compute separate scores for foreground and background corpus, or to simply pool the counts. Details for use of this feature are available in the *Moses* online documentation.⁴

5. Building and using sampling phrase tables in *Moses*

5.1. *Moses* compilation

Compile *Moses* as usual, but with the switch `--with-mm`:⁵

```
./bjam --with-mm --prefix=...
```

⁴<http://www.statmt.org/moses>

⁵Suffix-array based sampling phrase tables are scheduled to be included with the standard build soon.

The binaries `mtt-build`, `symal2mam`, and `mmlex-build` will be placed in the same directory as the `moses` executable.

5.2. Binarizing the word-aligned parallel corpus

Binarisation converts the corpus from a text representation into large arrays of 32-bit word IDs, and creates the suffix arrays. Word alignment information is also converted from a text representation (`symal` output format) to a binary format. In addition, a probabilistic word translation lexicon is extracted from the word-aligned corpus and also stored in binary format. All files are designed to be mapped directly in to memory for fast loading.

Let *corpus* be the base name of the parallel corpus. The tags *src* and *trg* are language tags that identify the source and the target language. Normally, these tags are mnemonic tags such as *en*, *fr*, *de*, etc.

- *corpus.src* is the source side of the corpus (one sentence per line, tokenized);
- *corpus.trg* is the respective target side in the same format;
- *corpus.src-trg.symal* is the word alignment between the two in the format produced by the `symal` word alignment symmetriser;
- */some/path/* is the path where the binarized model files will be stored. It must exist prior to running the binarizers. The path specification may include a file prefix *bname*. for the individual file names, in which case */some/path/bname*. should be used instead of */some/path/* in all steps.

Binarisation consists of four steps, the first three of which can be run in parallel. The

Step 1: binarise source side: `mtt-build < corpus.src -i -o /some/path/src`

Step 2: binarise target side: `mtt-build < corpus.trg -i -o /some/path/trg`

Step 3: binarise word alignments

`symal2mam < corpus.src-trg.symal /some/path/src-trg.mam`

Step 4: produce a word lexicon for lexical scoring

`mmlex-build corpus. src trg -o /some/path/src-trg.lex`

Steps 1 and 2 will produce 3 files each: a map from word strings to word IDs and vice versa (*.tdx*), a file with the binarized corpus (*.mct*), and the corresponding suffix array (*.sfa*). Steps 3 and 4 produce one file each (*.mam* and *.lex*, respectively).

5.3. Setting up entries in the `moses.ini` file

In the section `[feature]`, add the following two entries.

`LexicalReordering name=DM0 type=hier-mslr-bidirectional-fe-allff`

`Mmsapt name=PT0 lrfunc=DM0 path=/some/path/ L1=src L2=trg sample=1000`

Note that the value of the `path` parameter must end in `'/'` or `'.'`, depending on whether it points to a directory or includes a file name prefix. The value of the parameter `lrfunc` must match the name of the lexical reordering feature.

5.4. Setting up sampling phrase tables in EMS

In *Moses's Experiment Management System* (EMS), the use of sampling phrase tables can be specified by adding the following two lines to the EMS configuration file.

```
mmsapt = "sample=1000"
binarize-all = $moses-script-dir/training/binarize-model.perl
```

6. Configuring the phrase table

The phrase table implementation offers numerous configuration options. Due to space constraints, we list only the most important ones here; the full documentation can be found in the online *Moses* documentation at <http://statmt.org/moses>. All options can be specified in the phrase table's configuration line in `moses.ini` in the format `key=value`. Below, the letter '*n*' designates numbers in \mathbb{N} , '*f*' floating point numbers, and '*s*' strings.

6.1. General Options

sample=*n* the maximum number of samples considered per source phrase.

smooth=*f* the "smoothing" parameter. A value of 0.01 corresponds to a 99% confidence interval.

workers=*n* the degree of parallelism for sampling. By default (`workers=0`), all available cores are used. The phrase table implements its own thread pool; the general *Moses* option `threads` has no effect here.

cache=*n* size of the cache. Once the limit is reached, the least recently used entries are dropped first.

ttable-limit=*n* maximum number of distinct translations to return.

extra=*s* path to additional word-aligned parallel data to seed the foreground corpus for use in an interactive *dynamic* scenario where phrase tables can be updated while the server is running. This use case is explained in more detail in Germann (2014).

6.2. Feature Functions

Currently, word-level lexical translation scores are always computed and provided. Below we list some core feature scores that the phrase table can provide. A comprehensive list including experimental features is provided online at <http://statmt.org/moses>.

pfwd=g[+] forward phrase-level translation probability. If `g+` is specified, scores are computed and reported separately for the static background and the dynamic foreground corpus. Otherwise, the underlying counts are pooled.

pbwd=g[+] backwards phrase-level translation probability with the same interpretation of the value specified as for `pfwd`.

| source | sentence pairs | French tokens | English tokens |
|----------------------------------|----------------|----------------|----------------|
| CommonCrawl | 3.2 M | 86 M | 78 M |
| EuroParl | 2.0 M | 58 M | 52 M |
| Fr–En Gigaword | 21.4 M | 678 M | 562 M |
| News Commentary | 0.2 M | 6 M | 5 M |
| UN | 12.3 M | 367 M | 318 M |
| Total for TM training | 39.1 M | 1,185 M | 1,016 M |
| News data for LM training | 140.0 M | | 2,874 M |

Table 2. Corpus statistics for the parallel WMT-2015 French-English training data.

lenrat= $\{0|1\}$ phrase length ratio score (off/on). Phrase pair creation is modelled as a Bernoulli process with a biased coin: ‘heads’: produce a word in $L1$, ‘tails’: produce a word in $L2$. The bias of the coin is determined by the ratio of the lengths (in words) of the two sides of the training corpus. This score is the log of the probability that the phrase length ratio is no more extreme (removed from the mean) than observed.

rare= f rarity penalty: $\frac{f}{f+j}$, where j is the phrase pair joint count. This feature is always computed on the pooled counts of foreground and background corpus.

prov= f provenance reward: $\frac{j}{f+j}$. This feature is always computed separately for foreground and background corpus.

7. Performance on a large dataset

Table 3 shows build times and translation performance of two systems built with the large French–English data set available for the WMT-2015 shared translation task (cf. Tab. 2). The first system uses a pruned conventional phrase table binarized as a compact phrase table (Junczys-Dowmunt, 2012) (tuning was performed with an unpruned, filtered in-memory phrase table); the other system uses a sampling phrase table. The systems were tuned on 760 sentence pairs from the `newsdiscussdev2015` development set and evaluated on the `newsdiscusstest2015` test set.

For technical reasons, we were not able to run the **build processes** on dedicated machines with identical specifications; build times reported are therefore only approximate numbers. To give the conventional phrase table construction process the benefit of the doubt, the data binarization for the sampling phrase table was performed on a less powerful machine (8 cores) than conventional phrase table construction (24–32 cores), although not all steps in the process can utilize multiple cores. Nevertheless, even under these slightly unfair conditions the time savings of the sampling approach are obvious. The **translation speed** experiments were performed with cube pruning (pop limit: 1000) on the same 24-core machine with 148GB of memory, translating the test set of 1500 sentences (30,000 words) in bulk using all available cores on

| | conventional system | | | sampling phrase tables | | |
|--|---|-------------|--|---|-------------|--|
| phrase table build time | ≥ 20 hrs. | | | ca. 1h 30m | | |
| Model features: | total: 28 | | | total: 18 | | |
| • word penalty | yes | | | yes | | |
| • phrase penalty | yes | | | yes | | |
| • distortion distance | yes | | | yes | | |
| • language model | 5-gram Markov model | | | 5-gram Markov model | | |
| • TM: phrase transl. | forward, backward w/ Good-Turing sm. | | | forward, backward lower bound of 99% conf. interv. | | |
| • TM: lexical transl. | forward, backward | | | forward, backward | | |
| • rare counts | 6 bins: 1/2/3/4/6/10 | | | rarity penalty | | |
| • lex. reord. model | hierarchical-fe-mslr-all-ff | | | hierarchical-fe-mslr-all-ff | | |
| • phrase length ratio | no | | | yes | | |
| Evaluation (newsdiscusstest2015) | 3 independent tuning runs | | | | | |
| | run | BLEU | 95% conf. interval via bootstrap resampling | run | BLEU | 95% conf. interval via bootstrap resampling |
| batch MIRA | #1 | 33.16 | 32.07 – 34.21 | #1 | 33.16 | 31.96 – 34.27 |
| | #2 | 33.42 | 32.42 – 34.52 | #2 | 32.89 | 31.74 – 34.04 |
| | #3 | 33.30 | 32.16 – 34.39 | #3 | 33.12 | 32.03 – 34.20 |
| MERT | #1 | 32.19 | 31.15 – 33.13 | #1 | 34.25 | 33.11 – 35.37 |
| | #2 | 32.93 | 31.90 – 34.08 | #2 | 34.11 | 32.91 – 35.37 |
| | #3 | 31.53 | 30.39 – 32.68 | #3 | 33.80 | 32.69 – 34.90 |
| translation speed | | unpruned | top30 | sample=1000 | sample=100 | |
| | threads | 8 | 24 | 24 | 24 | |
| | wrds./sec. (sec./wrđ.) | 13 (0.075) | 547 (0.002) | 300 (0.003) | 501 (0.002) | |
| | snts./sec. (sec./snt.) | 0.7 (1.498) | 27 (0.037) | 15 (0.067) | 25 (0.040) | |
| | BLEU (best system) | 33.42 | 33.55 | 34.25 | 33.82 | |

Table 3. Features used and translation performance for the WMT15 fr-en experiments.

the machine. Prior to the start of *Moses*, all model files were copied to `/dev/null` to push them into the operating system’s file cache. Due to race conditions between threads, we limited the number of threads to 8 for the legacy system with the unpruned phrase table.

Notice that in terms of BLEU scores, the two systems perform differently with different tuning methods. The lower performance of MERT for the conventional system with 28 features is not surprising: it is well known that MERT tends to perform poorly when the number of features exceeds 20. That MIRA fares worse than MERT for the sampling phrase tables may be due to a sub-optimal choice of MIRA’s meta-parameters (cf. Hasler et al., 2011 for details on MIRA’s meta-parameters).

8. Conclusion


We have presented an efficient implementation of sampling phrase tables in *Moses*. With the recent integration of hierarchical lexicalized reordering models into the approach, sampling phrase tables reach the same level of translation quality while approaching *CompactPT* in terms of speed. In addition, sampling phrase tables offer the following advantages that make them an attractive option both for experimentation and research, and for use in production environments:

- They are much faster to build.
- They offer flexibility in the choice of feature functions used. Feature functions can be added or disabled without creating the need to re-run the entire phrase table construction pipeline.
- They have a lower memory footprint. It is not necessary to filter or prune the phrase tables prior to translation.

9. Availability

Sampling phrase tables are included in the master branch of *Moses* in the *Moses* github repository at <http://github.com/moses-smt/mosesdecoder.git>.

Acknowledgements

 This work was supported by the European Union's *Horizon 2020* research and innovation programme (H2020) under grant agreements 645487 (MMT) and 645452 (QT21). It extends work supported by the EU's *Framework 7* (FP7) program under grant agreements 287688 (MateCat), 287576 (CasMaCat), and 287576 (ACCEPT).

Bibliography

- Callison-Burch, Chris, Colin Bannard, and Josh Schroeder. Scaling Phrase-Based Statistical Machine Translation to Larger Corpora and Longer Phrases. In *43rd Annual Meeting of the Association for Computational Linguistics (ACL '05)*, pages 255–262, Ann Arbor, Michigan, 2005.
- Chiang, David. Hierarchical Phrase-Based Translation. *Computational Linguistics*, 33(2):1–28, 2007.
- Clopper, C.J. and E.S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 1934.
- Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12, Uppsala, Sweden, 2010.

- Foster, George F., Roland Kuhn, and Howard Johnson. Phrasetable Smoothing for Statistical Machine Translation. In *EMNLP*, pages 53–61, 2006.
- Galley, Michel and Christopher D. Manning. A Simple and Effective Hierarchical Phrase Reordering Model. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 848–856, Honolulu, Hawaii, 2008.
- Germann, Ulrich. Dynamic Phrase Tables for Machine Translation in an Interactive Post-editing Scenario. In *Proceedings of the Workshop on Interactive and Adaptive Machine Translation*, pages 20–31, 2014.
- Hasler, Eva, Barry Haddow, and Philipp Koehn. Margin Infused Relaxed Algorithm for Moses. *The Prague Bulletin of Mathematical Linguistics*, 96:69–78, 2011.
- Junczys-Dowmunt, Marcin. Phrasal Rank-Encoding: Exploiting Phrase Redundancy and Translational Relations for Phrase Table Compression. *Prague Bull. Math. Linguistics*, 98: 63–74, 2012.
- Li, Zhifei, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Joshua: An Open Source Toolkit for Parsing-Based Machine Translation. In *Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, 2009.
- Lopez, Adam. Hierarchical Phrase-Based Translation with Suffix Arrays. In *EMNLP-CoNLL*, pages 976–985, 2007.
- Lopez, Adam. *Machine Translation by Pattern Matching*. PhD thesis, University of Maryland, College Park, MD, USA, 2008.
- Manber, Udi and Gene Myers. Suffix Arrays: A New Method for On-line String Searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '90*, pages 319–327, Philadelphia, PA, USA, 1990. ISBN 0-89871-251-3.
- Mausser, Arne, David Vilar, Gregor Leusch, Yuqi Zhang, and Hermann Ney. The RWTH Machine Translation System for IWSLT 2007. In *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, 2007.
- Och, Franz Josef. Minimum Error Rate Training in Statistical Machine Translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan, 2003.
- Schwartz, Lane and Chris Callison-Burch. Hierarchical Phrase-Based Grammar Extraction in Joshua: Suffix Arrays and Prefix Trees. *The Prague Bulletin of Mathematical Linguistics*, 93: 157–166, 2010.
- Zens, Richard and Hermann Ney. Efficient Phrase-Table Representation for Machine Translation with Applications to Online MT and Speech Translation. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '07)*, pages 492–499, Rochester, New York, 2007.

Address for correspondence:

Ulrich Germann

ugermann@inf.ed.ac.uk

University of Edinburgh • 10 Crichton Street • Edinburgh, EH8 9AB, United Kingdom



The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015 51-62

Grasp: Randomised Semiring Parsing

Wilker Aziz

Universiteit van Amsterdam

Abstract

We present a suite of algorithms for inference tasks over (finite and infinite) context-free sets. For generality and clarity, we have chosen the framework of *semiring parsing* with support to the most common semirings (e.g. FOREST, VITERBI, k-BEST and INSIDE). We see parsing from the more general viewpoint of weighted deduction allowing for arbitrary weighted finite-state input and provide implementations of both bottom-up (CKY-inspired) and top-down (EARLEY-inspired) algorithms. We focus on approximate inference by Monte Carlo methods and provide implementations of *ancestral sampling* and *slice sampling*. In principle, sampling methods can deal with models whose independence assumptions are weaker than what is feasible by standard dynamic programming. We envision applications such as monolingual constituency parsing, synchronous parsing, context-free models of reordering for machine translation, and machine translation decoding.

1. Introduction

Many inference tasks in Natural Language Processing (NLP) involve operations over weighted context-free sets of solutions. Typical examples are constituency parsing and hierarchical Statistical Machine Translation (SMT). These weighted sets represent functions over large spaces of tree-structured solutions. We focus on cases where these functions have a probabilistic interpretation and can be represented by a weighted Context-Free Grammar (CFG). Common inference tasks involve finding the solution which maximises the underlying function (*optimisation*), or the one which minimises an expected loss (*minimum Bayes risk*, MBR), or the one that is marginally optimum. Depending on the complexity of the underlying distribution, these *decision rules* can be hard to compute. Particularly, in its most general form, the marginalisation problem is NP-complete (Sima'an, 1996). The MBR objective can also become

unwieldy depending on the complexity of the loss function. To all such tasks, Monte Carlo (MC) methods provide sound approximations (often with some guarantees) based on random simulation techniques. We investigate and propose a framework towards a general and flexible MC approach to inference with weighted CFGs.

2. Semiring parsing

A first step in generalising parsing is understanding it as the intersection between the language of a grammar and that of a Finite-State Automaton (FSA) (Bar-Hillel et al., 1961; Billot and Lang, 1989). This insight motivates a generalisation of parsing to arbitrary weighted finite-state input (Nederhof and Satta, 2003; Dyer and Resnik, 2010). In this context, not only the set intersection is computed, but strings in the resulting set are weighted by the product of their input weights. Parsing (or *intersecting*) an automaton finds use in applications where it is natural to represent uncertainty over finite-state input, for instance due to automatic pre-processing.

Another general view of parsing, orthogonal to the previous point, arises from the relationship between parsing and logic deductive systems (Pereira and Warren, 1983). Specifying a parser by means of a deductive system abstracts away from implementation details redirecting attention back to the parsing strategy itself. Shieber et al. (1995) lay down the principles of parsing as deduction and offer an extensive discussion regarding implementation. Arguably their most exciting result is to uncover the relationship between parsers for different grammar formalisms (context-free and beyond) which turn out to share the same (or very closely related) set of deduction rules under perhaps different control mechanisms.

In deductive parsing, a *deduction rule* is a template $\frac{A_1 \dots A_k}{B} C_1 \dots C_j$, where $A_1 \dots A_k$ (called *antecedents*), $C_1 \dots C_j$ (called *side condition*) and B (called *consequent*) are *items*. A rule states that, if the side condition holds and the antecedents have already been inferred, we can infer the consequent. Each possible way to deduce an item from the grammar rules by instantiation of deduction rules is called an *item derivation* and denoted by D . Any given instance x of an item implicitly defines a set D_x of possible ways to infer x . To test membership, the parser needs to prove at least one item derivation (a sequence of instantiation of deduction rules) that leads to a *goal* item form, or to show that no such sequence exists. A parse *forest* can be thought of as the exhaustive instantiation of all item derivations compatible with the grammar and the input.¹

On top of this item-based description, parsing can be further generalised to comply with an algebraic view of formal languages (Goodman, 1999). In this view, a parser does not simply test for membership, instead, it performs abstract computa-

¹The word **exhaustive** here does not imply inefficiency. In fact, item derivations are defined recursively which leads to efficient representation. Suppose $\frac{a_1 \dots a_k}{b} c_1 \dots c_j$ an instantiation of a deduction rule and $D_{a_1} \dots D_{a_k}$ sets of item derivations deducing $a_1 \dots a_k$, then $(b : D_{a_1}, \dots, D_{a_k})$ is itself an item derivation. This is very similar to the use of *back-pointers* in chart parsing (Billot and Lang, 1989).

tions under a given *semiring*.² Goodman (1999) combines the representational power of deductive systems with the generality of semirings to describe, under a uniform and simple representation, parsers that perform a multitude of tasks from recognition (BOOLEAN semiring) to best-first enumeration (k-BEST semiring).

In this work, we provide implementation of semiring parsing for CFGs and the following semirings: BOOLEAN and COUNTING for recognition and counting; INSIDE used to compute the probability of a string; VITERBI used to compute the probability of the best derivation; 1-BEST and k-BEST used to enumerate derivations in best-first order; and FOREST used to represent the set of all derivations in some compact manner.

We deploy semiring parsing as deductive programs which intersect an epsilon-free weighted CFG with a weighted deterministic FSA. Because CFGs are closed under intersection with automata (Hopcroft and Ullman, 1979), the result is another CFG. We observe from the state of the deductive program’s execution a parse forest which we represent using a hypergraph (Gallo et al., 1993).³ This is basically our implementation of the FOREST semiring. Computing values in the other semirings follows by direct application of the value recursion (Goodman, 1999, Equations 5 and 7) using the corresponding semiring operators (Goodman, 1999, Figure 5). In solving the value recursion, we consult the forest as a data structure which compactly organises items and their derivations.

We present GRASP, a toolkit which incorporates these ideas while facilitating research on sampling methods for structured prediction problems, particularly, parsing and machine translation. In the remainder of this paper we present its design as well as SAMPLE and SLICE, two novel semirings which play a central role in delivering sampling algorithms for complex distributions over context-free sets of solutions.

3. Randomised semiring parsing

Our goal is to sample probabilistically from a distribution proportional to $f(\mathbf{d})$ defined over a support $\mathcal{D}_G(\mathbf{x})$, where \mathbf{x} is an input object and $\mathcal{D}_G(\mathbf{x})$ is the derivation forest resulting of intersecting G (a CFG) and an FSA based on \mathbf{x} . Moreover, we assume f to factorise as shown in Equation 1, where $\psi(\mathbf{d})$ is some non-negative function of \mathbf{d} which is not assumed to factorise further (we call ψ a *nonlocal* parameterisation), and $\theta(\mathbf{d})$ is some non-negative function of \mathbf{d} compliant with the independence assumptions of G which underlie $\mathcal{D}_G(\mathbf{x})$ (we call θ a *local* parameterisation). Equation

²A semiring is an algebraic structure with operations that generalise the arithmetic addition and multiplication. The generalisation extends beyond numerical objects and may be defined over arbitrary sets (e.g. the tree-language of a context-free grammar).

³A hypergraph, more specifically a *backward-hypergraph*, is a generalisation of a graph where an edge connects a sequence of tail nodes under a head node and carries a weight. The nodes can be thought of as grouping item derivations, an edge can be thought of as an instantiated deduction rule. Hypergraphs are popular data structures in parsing (Klein and Manning, 2005) and MT (Huang, 2008).

2 illustrates how $\theta(\mathbf{d})$ factorises in terms of the rules used in the derivation \mathbf{d} , where r_s is a rule headed by s and θ_{r_s} is the rule's value.

$$f(\mathbf{d}) = \psi(\mathbf{d}) \times \theta(\mathbf{d}) \quad (1)$$

$$= \psi(\mathbf{d}) \times \prod_{r_s \in \mathbf{d}} \theta_{r_s} \quad (2)$$

The nonlocal dependencies introduced by $\psi(\mathbf{d})$ make inference over $f(\mathbf{d})$ a very difficult task. In order to make inference feasible, we resort to the principle of sampling by data augmentation (Tanner and Wong, 1987). In Data Augmentation (DA), we extend the target distribution with auxiliary variables \mathbf{u} that make sampling from the joint $f(\mathbf{d}, \mathbf{u})$ easier than sampling from the original distribution. Simulation then follows by Gibbs sampling, a Markov Chain Monte Carlo (MCMC) technique, whereby we sample each component in turn holding the remaining components fixed. That is, if we draw $\mathbf{u} \sim f(\mathbf{u}|\mathbf{d})$ followed by a draw $\mathbf{d} \sim f(\mathbf{d}|\mathbf{u})$, then the pair (\mathbf{u}, \mathbf{d}) is a draw from the joint distribution, and \mathbf{d} is a draw from the target.

The benefit of DA comes from the fact that we can choose the shape of the joint relatively flexibly, and we do so aiming at conditionals $f(\mathbf{u}|\mathbf{d})$ and $f(\mathbf{d}|\mathbf{u})$ that are easy to sample from. A common choice of joint for functions of the form shown in Equation 2 is illustrated in Equation 3,⁴ where $\delta_A(x)$ is the Dirac's measure which equals 1 iff $x \in A$. This principle underlies, and can be seen as a multivariate generalisation of, a univariate sampling technique called *slice sampling* (Neal, 2003). With such a choice of joint, we typically call the auxiliary variables *slice variables*. They define a "slice" of the distribution, i.e. a subset of $\mathcal{D}_G(\mathbf{x})$ for which $f(\mathbf{d}, \mathbf{u}) > 0$.

$$f(\mathbf{d}, \mathbf{u}) = \psi(\mathbf{d}) \times \prod_{s:r_s \in \mathbf{d}} \delta_{(0, \theta_{r_s})}(\mathbf{u}_s) \times \prod_{s:r_s \notin \mathbf{d}} \phi(\mathbf{u}_s; \boldsymbol{\alpha}) \quad (3)$$

First, we make the harmless independence assumption that $f(\mathbf{u}|\mathbf{d}) = \prod_{\mathbf{u}_s} f(\mathbf{u}_s|\mathbf{d})$, where the auxiliary random vector \mathbf{u} is indexed by items in the derivation forest (nodes in the hypergraph), and f factorises as a product of independent distributions (one for each item). Then, we specify the conditional $f(\mathbf{u}_s|\mathbf{d})$ as shown in Equation 4, where ϕ is a probability distribution with parameters $\boldsymbol{\alpha}$. Basically, the conditional is a uniform distribution when the item associated with \mathbf{u}_s participates in the conditioning derivation through rule r_s , and it falls back to a given less informed distribution ϕ otherwise.

$$f(\mathbf{u}_s|\mathbf{d}) = \begin{cases} \frac{\delta_{(0, \theta_{r_s})}(\mathbf{u}_s)}{\theta_{r_s}} & \text{if } r_s \in \mathbf{d} \\ \phi(\mathbf{u}_s; \boldsymbol{\alpha}) & \text{otherwise} \end{cases} \quad (4)$$

⁴Errata: after publication, we noticed that we had misrepresented $f(\mathbf{d}, \mathbf{u})$ in Equation 3, a mistake we remedy in this electronic version.

Blunsom and Cohn (2010) devised a special case of this slice sampling technique for binary Inversion Transduction Grammars (ITGs) in the context of Bayesian grammar induction for translation. In their work, s represents a pair of aligned spans, $0 \leq \theta_{r_s} \leq 1$ with $\sum_{r_s} \theta_{r_s} = 1$ for any given s are the parameters of a probabilistic ITG, and there is no $\psi(\mathbf{d})$ (otherwise thought of as $\psi(\mathbf{d}) = 1$). Thus they choose $\phi(u_s; \boldsymbol{\alpha}) = \text{Beta}(u_s; a, b)$, a Beta distribution with shape parameters a and b . We generalise their work to arbitrary weighted CFGs including those parameterised by log-linear models as common in SMT. To do so, we extend s to correspond to an arbitrary item in the derivation forest program, and we choose ϕ in different ways.⁵

The final step in the design of our sampler is to derive what $f(\mathbf{d}|\mathbf{u})$ turns out to be under the conditions above. Equations 5 to 7 are crucial, particularly, the latter implies that every derivation for which some $\theta_{r_s} \leq u_s$ will have zero probability regardless of the assignments of other slice variables. This basically means that $f(\mathbf{d}|\mathbf{u})$ can be represented by a “truncated” derivation forest such that $u_s < \theta_{r_s}$ for every r_s . We can interpret the slice variables as random thresholds on the values associated with item derivations of a given item.

$$f(\mathbf{d}|\mathbf{u}) \propto f(\mathbf{d}) \times f(\mathbf{u}|\mathbf{d}) \quad (5)$$

$$= \psi(\mathbf{d}) \times \prod_{r_s} \theta_{r_s} \times \prod_{u_s: r_s \in \mathbf{d}} \frac{\delta_{(0, \theta_{r_s})}(u_s)}{\theta_{r_s}} \times \prod_{u_s: r_s \notin \mathbf{d}} \phi(u_s; \boldsymbol{\alpha}) \quad (6)$$

$$\propto \psi(\mathbf{d}) \times \prod_{r_s} \frac{\delta_{(0, \theta_{r_s})}(u_s)}{\phi(u_s; \boldsymbol{\alpha})} \quad (7)$$

Obviously, because $\psi(\mathbf{d})$ breaks context-free independence assumptions, sampling from $f(\mathbf{d}|\mathbf{u})$ (Equation 7) might remain challenging. On the one hand, if $\psi(\mathbf{d})$ makes a low-order Markov assumption, exactly rescoring the slice might be possible, particularly, if slices are sufficiently thin. On the other hand, thin slices reduce the mobility of the sampler increasing autocorrelation. Moreover, if slices are deterministically too thin, we may risk trapping the sampler to a subset of the state space, which would compromise ergodicity. A more general solution is not to assume that we can sample from $\psi(\mathbf{d})$ exactly, and rely on MCMC techniques instead. One possible technique is to uniformly sample a subset (of some predetermined size) of the slice, and evaluate Equation 7 only for the derivations in that subset.

Finally, we note a connection to semiring parsing. We can define a `SAMPLE` semiring which is to the `INSIDE` semiring as `1-BEST` is to `VITERBI`. Informally, by modifying the `1-BEST` addition operation so that it samples from the distribution associated with the `INSIDE` values of item derivations (instead of maximising over their `VITERBI` values), we

⁵For example, for log-linear models $\phi(u_s; \boldsymbol{\alpha}) = \text{Exp}(u_s, \lambda)$, an exponential distribution with rate parameter λ . Note that the greater λ , the thicker the slice (the mean of the exponential distribution is λ^{-1}).

```

1 # Create and source a dedicated virtual environment for Grasp
2 ~$ virtualenv -p python3 ~/envs/grasp; source ~/envs/grasp/bin/activate
3 # Clone and install kenlm (if you intend to use Grasp's decoder)
4 (grasp)~$ git clone https://github.com/kpu/kenlm.git; cd kenlm
5 (grasp)~$ python setup.py install; cd ..
6 # Install general dependencies
7 (grasp)~/grasp$ pip install numpy scipy cython
8 # Clone and install Grasp
9 (grasp)~$ git clone https://github.com/wilkeraziz/grasp.git; cd grasp
10 (grasp)~/grasp$ python setup.py install
11 # A better option for contributors is: python setup.py develop

```

Figure 1. Installing Grasp.

get an independent random sample. Similarly, we can define a SLICE semiring which is analogous to the FOREST semiring, however, it produces a sliced forest. SLICE conditions on random assignments of an auxiliary vector based on a previous sample.⁶

4. GRASP

Our toolkit is available on github <https://github.com/wilkeraziz/grasp> under Apache 2.0 license. It is written in *python3* and *cython*, it builds trivially with *setuptools*, and it has very few (rather standard) dependencies. GRASP is primarily developed for UNIX-based systems (e.g. Linux and OS X). Figure 1 illustrates how to install GRASP using a dedicated *virtual environment* running *python3*.

GRASP comes with a CFG parser and a hierarchical SMT decoder. Figure 2 illustrates how to run the parser in two different modes. The first command illustrates exact inference (which requires complete parse forests) and outputs the VITERBI derivation, the 100-best derivations (`--kbest 100`), and 1000 independent samples (`--samples 1000`). The second command illustrates slice sampling, where $\phi(u_s; \alpha)$ defaults to $\text{Beta}(u_s; 0.1, 1)$. In both cases, `python -m grasp.cfg.parser` invokes the parser; `wsj00` is an example grammar shipped with GRASP, and output specifies a directory where output files are stored. The last command illustrates the output directory structure: it contains the independent ancestral samples, the k-best derivations, the Viterbi derivation, the slice samples and a configuration file which documents the experiment (`args.ini`).⁷

⁶An initial derivation can be obtained by sampling from the local model alone. If the grammar is too large, an initial sample can be obtained from a smaller grammar sharing a subset of the nonterminals of the original one (or where a coarse-to-fine mapping exists).

⁷A complete list of features and options can be found at <https://github.com/wilkeraziz/grasp/blob/master/grasp/cfg/README.md>.


```

1 # Framework: exact inference
2 (grasp)~/grasp/examples/ptb$ echo -e 'I was given a million dollars .' | \
3 python -m grasp.cfg.parser wsj00 output --grammarfmt discodop --start TOP \
4 --unkmodel stfd6 --log --viterbi --kbest 100 --samples 1000 --experiment mtm -v
5 # Framework: slice sampling
6 (grasp)~/grasp/examples/ptb$ echo -e 'I was given a million dollars .' | \
7 python -m grasp.cfg.parser wsj00 output --grammarfmt discodop --start TOP \
8 --unkmodel stfd6 --log --samples 1000 --framework slice --experiment mtm -v
9 # Output directory
10 (grasp)~/grasp/examples/ptb$ tree --charset=ascii output/mtm/
11 output/mtm/
12 |-- ancestral          |-- args.ini          |-- slice
13 | |-- derivations     |-- kbest            | |-- derivations
14 | | |-- 0.gz          | |-- 0.gz           | | |-- 0.gz
15 | |-- trees           |-- viterbi          | |-- trees
16 | | |-- 0.gz          | |-- 0.gz           | | |-- 0.gz

```

Figure 2. Example run of Grasp’s CFG parser.

Figure 3 illustrates how to run the hierarchical decoder. The first command illustrates exact inference. Note that with exhaustive forest rescoring, we cannot go beyond a bigram language model and very short sentences. The second command illustrates sliced rescoring: $\phi(u_s; \alpha)$ is set to $\text{Exp}(u_s; 1)$ (`--prior const 1`), each slice is uniformly subsampled (`--within uniform`) making small fixed-size batches (`--batch 100`) for rescoring (Equation 7). In both cases, `python -m grasp.cfg.decoder` invokes the decoder; synchronous grammars are stored in `grammars`; `--weights` specify model weights for model components such as rule table (`--rt`), word penalty (`--wp`), arity penalty (`--ap`), and language model (`--lm`). The output directory structure is very similar to the one produced by the parser.⁸ Output files are sorted lists of hypotheses (best-first), as the last command illustrates.

In this section we also provide some preliminary experiments with our proposed sampler in the context of decoding for hierarchical phrase-based (hiero) models (Chiang, 2005). We report on experiments conducted using the BTEC Chinese-English corpus (Takezawa et al., 2002). Grammar extraction follows the approach of Lopez (2007), we use the implementation of Baltescu and Blunsom (2014). We use `cdec` (Dyer et al., 2010) to train a linear model with MIRA (Cherry and Foster, 2012).⁹ We trained mod-

⁸A complete list of features and options can be found at <https://github.com/wilkeraziz/grasp/blob/master/grasp/mt/README.md>.

⁹Such models are not inherently probabilistic, thus we need to artificially scale the parameters returned by the optimiser. In the experiments in this paper, we multiplied the weights by 10 (`--temperature 0.1`).

```

1 # Framework: exact inference
2 (grasp)~/grasp/examples/mt$ head -n1 input | python -m grasp.mt.decoder output \
3 --grammars grammars --glue-grammar glue --pass-through \
4 --rt --wp WordPenalty -0.43429466 --ap Arity -0.43429466 \
5 --lm LanguageModel 2 btec.klm2 --weights mira/lm2-p \
6 --temperature 0.1 --viterbi --kbest 100 --samples 1000 --experiment mtm -v
7 # Framework: slice sampling
8 (grasp)~/grasp/examples/mt$ head -n1 input | python -m grasp.mt.decoder output \
9 --grammars grammars --glue-grammar glue --pass-through \
10 --rt --wp WordPenalty -0.43429466 --ap Arity -0.43429466 \
11 --lm LanguageModel 3 btec.klm3 --weights mira/lm3-p \
12 --temperature 0.1 --temperature0 10 --samples 1000 --framework slice \
13 --batch 100 --within uniform --prior const 1 --experiment mtm -v
14 (grasp)~/grasp/examples/mt$ zcat < output/mtm/slice/yields/0.gz
15 # MCMC samples=1000
16 # estimate          count  derivations  yield
17 0.601              601    5            a throbbing pain .
18 0.271              271    5            throbbing pain .
19 0.065              65     4            is a throbbing pain .
20 0.032              32     4            is throbbing pain .
21 0.027              27     2            throbbing pain in my temples .
22 0.002              2      1            throbbing pain in pain .
23 0.001              1      1            throbbing pain the pain .
24 0.001              1      1            throbbing pain hurts my .

```

Figure 3. Example run of Grasp’s hierarchical SMT decoder.

els using standard features as well as a bigram/trigram Language Model (LM) component. Language modelling and LM queries are done with Implz/kenlm (Heafield, 2011; Heafield et al., 2013). Additionally, we trained two locally parameterised models whose LMs were made *stateless*, i.e. n-grams are scored using as much context as available within translation rules, but context information (LM *state*) is discarded at the boundaries of nonterminals. Finally, as a decision rule, GRASP uses an approximation to MBR known as *consensus decoding* (DeNero et al., 2009) based on an empirical distribution estimated from 2,000 samples.

We report BLEU scores (Papineni et al., 2002) from *multibleu*, an implementation distributed with Moses (Koehn et al., 2007). Results are averaged over 3 runs for a random subset comprising of 20% of BTEC’s development set.¹⁰ Table 1 compares GRASP’s performance to cdec’s. The stateless models can be decoded exactly by both cdec (which uses a Viterbi decision rule) and GRASP (which applies consensus decod-

¹⁰The exact subset is available at the tool’s repository on github.

| | System | LM | BLEU |
|----|-------------------|--------------------|--------------|
| 1 | cdec | 2-gram (stateless) | 33.6 |
| 2 | GRASP | 2-gram (stateless) | 33.85 |
| 3 | GRASP (batch=100) | 2-gram | 36.59 ± 0.61 |
| 4 | GRASP (batch=200) | 2-gram | 37.47 ± 0.67 |
| 5 | cdec (pop=200) | 2-gram | 41.46 |
| 6 | cdec | 3-gram (stateless) | 33.92 |
| 7 | GRASP | 3-gram (stateless) | 33.9 |
| 8 | GRASP (batch=100) | 3-gram | 37.78 ± 0.64 |
| 9 | GRASP (batch=200) | 3-gram | 38.89 ± 0.67 |
| 10 | cdec (pop=200) | 3-gram | 46.33 |

Table 1. Slice sampler using $\phi(\mathbf{u}_s; \boldsymbol{\alpha}) = \text{Exp}(\mathbf{u}_s; \lambda = 1.0)$.

ing to 2,000 ancestral samples), thus both systems perform equally well (rows 1-2 and 6-7). For nonlocal models, cdec employs *cube pruning* (pop limit 200), and GRASP employs the proposed *slice sampling* procedure uniformly subsampling slices to produce batches of 100 or 200 samples. We can see that GRASP succeeds to incorporate the language model to a certain extent, it outperforms stateless models by about 5 BLEU points (rows 3-4 vs rows 1-2, and rows 8-9 vs rows 6-7). However, it still lags behind cube pruning (rows 5 and 10). By rescoreing larger batches from each slice (row 4 vs row 3, and row 9 vs row 8), we notice significant improvements, which indicates that sampling from $f(\mathbf{d}|\mathbf{u})$ is indeed a bottleneck.

In order to improve GRASP’s translation accuracy, we need to sample more efficiently from slices. The general aim is to increase the sampler’s mobility and reduce autocorrelation. Indeed, subsampling slices uniformly is a very simple strategy. We are currently investigating alternatives based on more sophisticated sampling techniques. In the experiments reported, GRASP draws on average 7.5 samples per second (with batch=100) and 4.7 samples per second (with batch=200). Note that to obtain a sample, GRASP must first obtain a slice (a random subset of the complete forest), subsample such slice reducing it to a fixed-size batch, rescore the batch producing an empirical distribution for $f(\mathbf{d}|\mathbf{u})$ (Equation 7), and finally, sample a derivation from this empirical distribution. To improve GRASP’s time performance we are translating to *cython* some of the core procedures associated with these steps.

Finally, Figure 4 lists the features GRASP currently supports (this set is growing at a fast pace!).

| | |
|-----------------------|---|
| Grammar formalism | epsilon-free CFGs |
| Weighted deduction | bottom-up (exact and sliced), top-down (exact and sliced) |
| Forest rescoring | top-down (exact and sliced) |
| Real-valued semirings | BOOLEAN, COUNTING, VITERBI, INSIDE |
| Value recursion | robust to cycles |
| Derivation semirings | 1-BEST, k-BEST, SAMPLE, FOREST, SLICE |
| Sampling algorithms | ancestral sampling, slice sampling |
| LM queries | kenlm |
| Applications | constituency parsing, decoding for hiero models |

Figure 4. Features in Grasp’s current release.

5. Conclusion

We have given a quick overview of semiring parsing and discussed a novel sampling technique for inference over complex structured spaces which is compatible with this general framework. We have presented GRASP, a toolkit for new directions in inference for applications such as parsing and machine translation. With this release, we hope to lower the initial implementation burden associated with complex structured state spaces, and we invite contributors to explore new sampling algorithms as well as new applications.

Acknowledgements

This research is funded by The Netherlands Organisation for Scientific Research (NWO), NWO VICI grant nr. 277-89-002.

Bibliography

- Baltescu, Paul and Phil Blunsom. A Fast and Simple Online Synchronous Context Free Grammar Extractor. *The Prague Bulletin of Mathematical Linguistics*, 102(1):17–26, October 2014.
- Bar-Hillel, Yehoshua, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, (14):143–172, 1961.
- Billot, Sylvie and Bernard Lang. The Structure of Shared Forests in Ambiguous Parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June 1989. Association for Computational Linguistics.
- Blunsom, Phil and Trevor Cohn. Inducing Synchronous Grammars with Slice Sampling. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 238–241, Los Angeles, California, June 2010. Association for Computational Linguistics.

- Cherry, Colin and George Foster. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 427–436, Montréal, Canada, June 2012. Association for Computational Linguistics.
- Chiang, David. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 263–270, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.
- DeNero, John, David Chiang, and Kevin Knight. Fast Consensus Decoding over Translation Forests. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 567–575, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
- Dyer, Chris and Philip Resnik. Context-free Reordering, Finite-state Translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, HLT '10*, pages 858–866, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
- Dyer, Chris, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik. cdec: a decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations, ACLDemos '10*, pages 7–12, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
- Gallo, Giorgio, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, Apr. 1993.
- Goodman, Joshua. Semiring parsing. *Computational Linguistics*, 25(4):573–605, Dec. 1999.
- Heafield, Kenneth. KenLM: faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation, WMT '11*, pages 187–197, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.
- Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.
- Hopcroft, John E. and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1979.
- Huang, Liang. Advanced Dynamic Programming in Semiring and Hypergraph Frameworks. In *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*, pages 1–18, Manchester, UK, August 2008. Coling 2008 Organizing Committee.
- Klein, Dan and Christopher D. Manning. Parsing and Hypergraphs. In Bunt, Harry, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*, pages 351–372. Springer Netherlands, 2005.

- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, ACL '07*, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
- Lopez, Adam. Hierarchical Phrase-Based Translation with Suffix Arrays. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 976–985, Prague, Czech Republic, June 2007. Association for Computational Linguistics.
- Neal, Radford M. Slice Sampling. *Annals of statistics*, 31:705–741, June 2003.
- Nederhof, Mark-Jan and Giorgio Satta. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, Nancy, France, April 2003.
- Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02*, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
- Pereira, Fernando C. N. and David H. D. Warren. Parsing As Deduction. In *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics, ACL '83*, pages 137–144, Stroudsburg, PA, USA, 1983. Association for Computational Linguistics.
- Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.
- Sima'an, Khalil. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th conference on Computational linguistics - Volume 2, COLING '96*, pages 1175–1180, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.
- Takezawa, Toshiyuki, Eiichiro Sumita, Fumiaki Sugaya, Hirofumi Yamamoto, and Seiichi Yamamoto. Toward a Broad-coverage Bilingual Corpus for Speech Translation of Travel Conversations in the Real World. In *Third International Conference on Language Resources and Evaluation, LREC, Las Palmas, Canary Islands - Spain, May 2002*. European Language Resources Association.
- Tanner, Martin A. and Wing Hung Wong. The Calculation of Posterior Distributions by Data Augmentation. *Journal of the American Statistical Association*, 82(398):528–540, June 1987.

Address for correspondence:

Wilker Aziz
w.aziz@uva.nl
Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107, F2.11
Netherlands

**MT-ComparEval: Graphical evaluation interface
for Machine Translation development**

Ondřej Klejch^a, Eleftherios Avramidis^b, Aljoscha Burchardt^b,
Martin Popel^a

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

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

Abstract

The tool described in this article has been designed to help MT developers by implementing a web-based graphical user interface that allows to systematically compare and evaluate various MT engines/experiments using comparative analysis via automatic measures and statistics. The evaluation panel provides graphs, tests for statistical significance and n-gram statistics. We also present a demo server <http://wmt.ufal.cz> with WMT14 and WMT15 translations.

1. Introduction

For language processing tasks like parsing or fact extraction, the expected results can be more or less clearly defined and it is comparably easy to assess the quality of a given system output. Due to the variation of language, ambiguity, etc. evaluating Machine Translation (MT) output can be almost as difficult as the translation itself. The evaluation methods and tools used in practice range from automatic measures that compare MT output against human reference translations via human error annotation up to usability studies. Even if we focus on automatic measures, we are confronted with several measures, options, and ways of measuring that have certain strengths and weaknesses in their diagnostic capacities and that often lead to different assessments of given systems or system variants to be compared. Therefore, often a mix of measures and eventual examination of random samples are used in devel-

opment cycles. However, the common practice of running many experiments over a period of time soon makes bookkeeping and tracing results challenging.

MT-ComparEval, the open-source tool described in this article has been designed in order to help MT developers by implementing a back-end evaluation system with a graphical user interface that allows comparing and evaluating different MT engines/experiments and settings through the use of several measures that represent the current best practice. The system helps to tie the development cycles together by linking three ingredients:

- An *evaluation panel*, which provides a graphical interface for comparing the performance of various systems on the same output, visualizing automatic scores and various types of statistics and manual checking of translation quality on a sentence level.
- A *back-end* that monitors some pre-defined storage locations for the addition of new translation outputs. Whenever a new translation output is detected, a new task is added in the background database.
- An *evaluation mechanism* that calculates MT evaluation scores based on a set of automatic evaluation metrics and statistical tests. These results are associated with the translation tasks in the database

The structure of this paper is as following: Section 2 compares our tool with previous work. Section 3 outlines the main functionality of the graphical interface. Section 4 describes the back-end and the evaluation mechanism and Section 5 provides information about the requirements, installation and implementation. Finally, Section 6 includes a summary and our aims for further improvements.

Demo server

In order to showcase the features of MT-ComparEval, we present a demonstration server <http://wmt.ufal.cz>. The interface includes pre-loaded system outputs from the WMT Shared Task 2014 and 2015 (Bojar et al., 2014, 2015).

2. Related Work

The state-of-the-art MT decoder Moses (Koehn et al., 2006) provides a similar evaluation interface as part of the **Experiment Management System** (Koehn, 2010). EMS is a set of scripts that automate and parallelize the full pipeline of training and testing SMT models with variable settings. On the one end of this pipeline, the developers have the possibility to display the testing results in a web-based interface. Comparing and contrasting our tool with EMS, we can see that:

- similar to our tool, the EMS interface organizes trained systems into groups of experiments. It includes a table for comparing the performance of different configurations of the same experiment, based on automatic scores, statistical signif-

icance tests, n-gram statistics and color-coded n-gram correctness markup for the output sentences.

- additional to our tool, EMS provides features due to its tight binding with the developed SMT systems, such as the phrase table coverage and the bilingual concordance analysis.
- contrary to EMS, which is tightly connected with the training pipeline and optimized for the Moses SMT scripts, our tool provides more flexibility, since the evaluation interface can be run independently of the production of the translation systems. The users can therefore import any translation, irrelevant of how it was constructed. Apart from SMT output, one can therefore import output by other types of systems, such as rule-based, dependency-based or hybrid systems.
- MT-ComparEval focuses on comparing two systems and shows their translations sorted by the difference in sentence-level BLEU (or another metric). Also the color highlighting shows the n-grams where one system is better than the other (while in EMS, the color corresponds to the length of the matching n-gram).

Similar evaluation panels are available through websites such as **Evaluation Matrix**,¹ which is used for displaying comparable scores for the translation system outputs participating in the translation shared task of the Workshop of Machine Translation (Callison-Burch et al., 2007). Commercial MT services, such as **Kantan** and **Asia Online**² also include graphical panels with automatic evaluations in their interface for training models. No system mentioned in this paragraph offers open-source code nor advanced sentence-level comparisons, to the best of our knowledge.

The concept of sentence-by-sentence evaluation is available within post-editing tools such as **PET** (Aziz et al., 2012) or human evaluation panels such as **Appraise** (Federmann, 2010), although these include no automatic evaluation scores and graphical comparison panels. **MT-EQuAl** (Girardi et al., 2014), a graphical tool for manual error annotation, allows for visualizing erroneous words and phrases that have been pre-computed by **Hjerson** (Popović, 2011), based on edit distance from the reference. Hjerson has been integrated also into **Addicter**³ (Zeman et al., 2011; Berka et al., 2012), which shows sentences (from the train or test corpus) with source-target alignment, manual or automatic word-level error classification and color highlighting and overall summaries.

¹<http://matrix.statmt.org>

² <https://www.kantanmt.com/> and <http://www.asiaonline.net>

³<https://wiki.ufal.ms.mff.cuni.cz/user:zeman:addicter>

MultEval⁴ (Clark et al., 2011) focuses on computing statistical significance using approximate randomization (Riezler and Maxwell, 2005) for three metrics: BLEU, TER and METEOR.

Similarities to our tool can also be seen in **Asiya** (Giménez and Màrquez, 2010), in the sense that it wraps the functionality of many automatic evaluation metrics in one program, although there is no graphical interface.

See MT-ComparEval wiki for a table of comparison with related tools.⁵

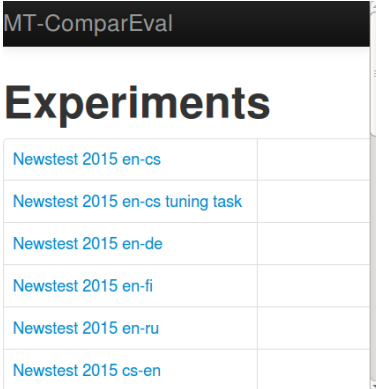
3. Evaluation Panel

Here, we present all basic screens and panes of the Evaluation panel, which is the graphical front-end of MT-ComparEval.

3.1. Listing of experiments and tasks

The start-up screen of the evaluation panel (Figure 1) provides a list of all the **experiments** that have been imported. An experiment consists of several **tasks** – variations of the same experiment with different settings. All tasks of the same experiment share the same source and reference translation and only the translated text varies. So a task can be a new version of a previously imported system, or a totally different system that nevertheless has been run on the same data.

Once the user selects an experiment, an evaluation table with all relevant tasks (Figure 2) is shown. The table contains (document-level) scores for each task entry calculated by (a selected subset of) the included automated metrics. The panel also includes a graphical representation to monitor the improvement of the scores among different versions of the same system.⁶ The metrics currently supported are Precision, Recall and F-score (all based on arithmetic average of 1-grams up to 4-grams) and BLEU (Papineni et al., 2002).⁷ MT-ComparEval computes both case-sensitive and



| MT-ComparEval | |
|---------------------------------|--|
| Experiments | |
| Newstest 2015 en-cs | |
| Newstest 2015 en-cs tuning task | |
| Newstest 2015 en-de | |
| Newstest 2015 en-fi | |
| Newstest 2015 en-ru | |
| Newstest 2015 cs-en | |

Figure 1. Part of the start-up screen of the evaluation panel with a list available experiments at wmt.ufal.cz.

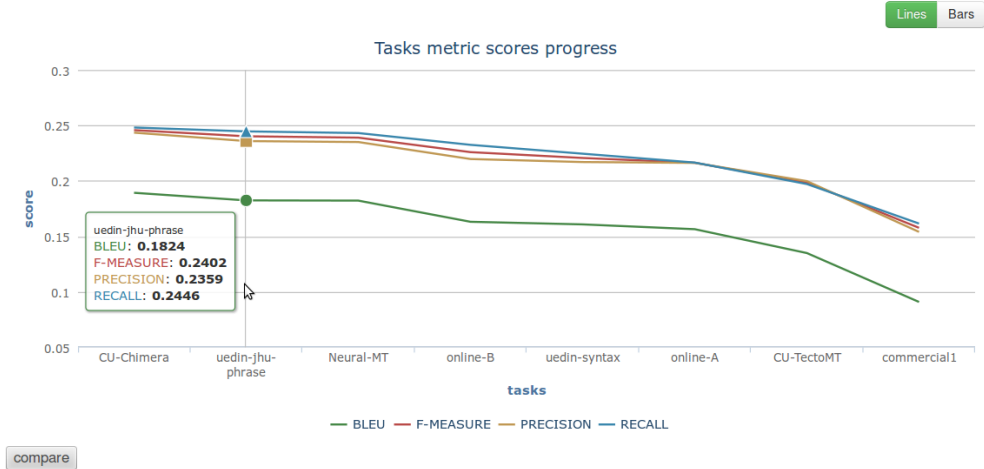
⁴<https://github.com/jhclark/multeval>

⁵ <https://github.com/choko/MT-ComparEval/wiki/Related-tools> (updates are welcome)

⁶ The line graph can be switched to a bar graph, which is more suitable for comparing unrelated systems.

⁷ BLEU uses a geometric mean of 1-grams up to 4-grams, so it needs smoothing for sentence-level scores. We have reimplemented the “official” BLEU script <ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13a.pl> with the option `--international-tokenization`.

Newstest 2015 en-cs



compare

| name | description | BLEU ↓ | F-MEASURE | PRECISION | RECALL | |
|---|-------------|--------|-----------|-----------|--------|----------------------|
| <input type="checkbox"/> CU-Chimera | | 0.1893 | 0.2458 | 0.2435 | 0.2481 | hide |
| <input type="checkbox"/> uedin-jhu-phrase | | 0.1824 | 0.2402 | 0.2359 | 0.2446 | hide |
| <input type="checkbox"/> Neural-MT | | 0.1822 | 0.239 | 0.2351 | 0.2431 | hide |

Figure 2. Listing of tasks of the “Newstest 2015 en-cs” experiment at wmt.ufal.cz.

case-insensitive (CIS) versions of the four metrics, but the current default setup is to show only case-sensitive versions in the Tasks screen. Users are able to turn the individual metrics on and off, to allow for easier comparisons.

Additionally, the users have the possibility to delete or hide particular tasks from this panel, e.g. in order to focus on a comparison between particular versions of their systems which show specific progress.

The name and the description of each task is editable,⁸ whereas the description text is collapsed by default to permit a better appearance of the table. The table can be re-sorted on demand, based on each automatic metric.

For inspection of the differences between two tasks (say, systemA and systemB), the user needs to mark the tasks’ checkboxes and click “Compare”. A new screen presents four panes: Sentences, Statistics, Confirmed n-grams and Unconfirmed n-grams, which are described in the following subsections.

⁸ Editing descriptions and deleting tasks and experiments is disabled at wmt.ufal.cz.

MT-ComparEval Newstest 2015 en-cs

Neural-MT CU-Chimera BLEU-cis

Sentences Statistics Confirmed n-grams Unconfirmed n-grams

Sentences

Options

| | | |
|--|--|--|
| <p>N-grams highlighting options</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Highlight confirmed n-grams <input checked="" type="checkbox"/> Highlight improving n-grams <input checked="" type="checkbox"/> Highlight worsening n-grams | <p>Diff highlighting options</p> <ul style="list-style-type: none"> <input type="checkbox"/> Show diff with reference <input checked="" type="radio"/> Show diff for Neural-MT <input type="radio"/> Show diff for CU-Chimera <input checked="" type="checkbox"/> Show diff with each other | <p>Sentences visibility options</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Show source <input checked="" type="checkbox"/> Show reference <input checked="" type="checkbox"/> Show Neural-MT <input checked="" type="checkbox"/> Show CU-Chimera <input checked="" type="checkbox"/> Show sentence level metrics |
|--|--|--|

| | | | | | | | | |
|-------------------|--|-----------------|------------------|----------------------|------------------|----------------------|---------------|-------------------|
| Source | The soundtrack is a mix of reggae and rap and the tunes are upbeat . | | | | | | | |
| Reference | Soundtrack je smésici reggae a rapu a melodie je optimistická . | | | | | | | |
| Neural-MT | The je smés , a rap a melodie jsou optimistická . | | | | | | | |
| CU-Chimera | Soundtrack je smésici reggae a rapu a melodie jsou optimistické . | | | | | | | |
| | BLEU | BLEU-cis | F-MEASURE | F-MEASURE-cis | PRECISION | PRECISION-cis | RECALL | RECALL-cis |
| Neural-MT | 0.1173 | 0.1173 | 0.1864 | 0.1864 | 0.1864 | 0.1864 | 0.1864 | 0.1864 |
| CU-Chimera | 0.6989 | 0.6989 | 0.7025 | 0.7025 | 0.7025 | 0.7025 | 0.7025 | 0.7025 |
| Diff | -0.5816 | -0.5816 | -0.5161 | -0.5161 | -0.5161 | -0.5161 | -0.5161 | -0.5161 |

Figure 3. Sentences pane with one sentence shown.

3.2. Sentences pane

This pane displays all sentences from the given testset translated by both the systems, one below the other, along with source sentence, reference sentence and scores (see Figure 3).⁹ The sentences are sorted according to the differences in the chosen sentence-level metric scores. This means that the sentences shown at the top are those, where systemB outperforms systemA the most. Such a view is very useful when checking for regressions of new versions of an MT system against a baseline or a previous version of the same system.

A set of checkboxes allow the user to highlight differences between the two systems in several ways:

- **Confirmed n-grams** are n-grams occurring both in the system output and in the reference.¹⁰ These are shown with light yellow (for systemA) and blue (for

⁹ The sentences are loaded lazily as needed when the user scrolls down.

¹⁰ If a given n-gram occurs e.g. three times in the system output and only twice in the reference, a heuristic algorithm (based on the longest common subsequence) is used to select two occurrences of the n-gram that will be marked as confirmed in the system output.

systemB) background. The confirmed n-grams are highlighted also in the reference, where light green color marks n-grams occurring in both system.

- **Improving n-grams** are confirmed n-grams occurring in only one of the systems. These are highlighted in the system outputs with darker yellow and blue.
- **Worsening n-grams** are unconfirmed n-grams (i.e. probably wrong translations) occurring in only one of the systems. These are highlighted with red.
- **Diff** of the reference and one of the systems: words in the longest common sub-sequence of the two sentences are underlined in green, other words in red.

Although multiple kinds of highlighting and sentences can be displayed simultaneously, users usually use checkbox options to enable only those they are currently interested in. One can for example hide everything except for the references with confirmed n-grams highlighted in light yellow, blue and green (for n-grams occurring in systemA only, systemB only and both systems, respectively). Highlighting the improving and worsening n-grams is also very useful because these are the “culprits” of the BLEU scores differences. Diff is useful e.g. for checking word-order differences (an n-gram may be confirmed, but not underlined in green because it is not in the same position as in the reference).

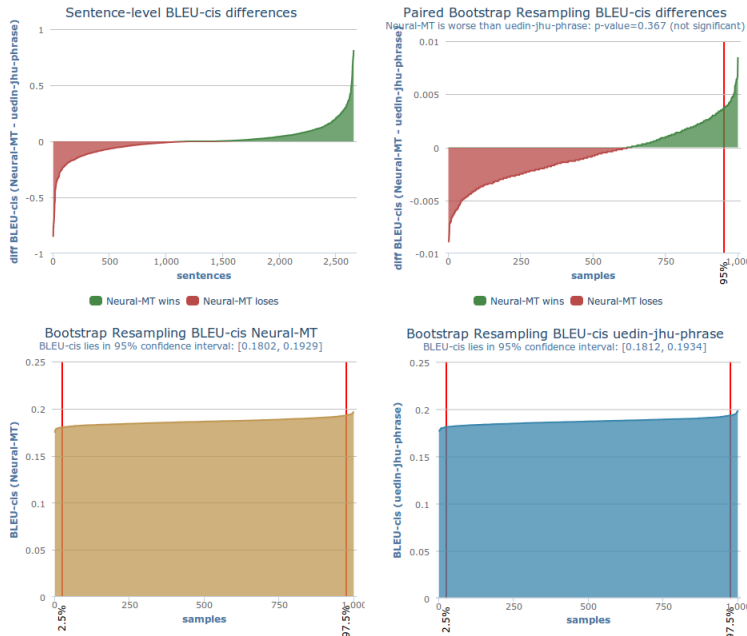


Figure 4. Evaluation panel for bootstrap resampling

3.3. Statistics pane (Bootstrap resampling)

This pane shows four area charts depicted at Figure 4 and all document-level metric scores (not depicted here). The bottom two charts show (*non-paired*) *bootstrap resampling* (Koehn, 2004) for systemA and systemB, respectively, to assess BLEU confidence intervals for the individual systems. The x-axis lists 1,000 resamples (generated from the original testset) sorted by their (document-level) BLEU, which is the y-axis.

The upper right chart shows *paired bootstrap resampling* (Koehn, 2004), where the x-axis lists 1,000 resamples and the y-axis is the difference in BLEU between systemA and systemB for the given resample. One-tailed p-value is in the chart header.

The upper left chart shows sentence-level BLEU difference (y-axis) for all the sentences in the testset (x-axis).¹¹

3.4. Confirmed and unconfirmed n-grams panes

MT-ComparEval counts how many times was an n-gram seen as *improving* for systemA (as defined in 3.2) in the whole testset. The top ten n-grams are shown in table “systemA wins” in the *Confirmed n-grams* pane (there are four such tables: for 1-grams up to 4-grams, but Figure 5 shows only the 1-grams). Similarly, the “systemB wins” table shows the top ten n-grams improving systemB. *Unconfirmed n-grams* work analogically for n-grams *worsening* systemA/B (the tables are labeled “systemA/B loses”). The user can click on any n-gram in the tables to see all sentences containing the n-gram (with the n-gram highlighted).

In near future, we would like to improve this pane to show the difference of current “systemA wins” and “systemB wins” scores for each n-gram, so the top 10 n-grams shown are more informative about the real problems and differences in the two systems.

n-grams confirmed by the reference

| | | 1-gram | |
|----|-----|----------------|-----------------|
| | | Neural-MT wins | CU-Chimera wins |
| . | 215 | . | 239 |
| se | 186 | * | 210 |
| * | 180 | na | 104 |
| na | 81 | . | 75 |
| v | 66 | v | 62 |
| to | 60 | se | 61 |
| ze | 52 | . | 50 |
| si | 48 | to | 40 |
| je | 46 | je | 30 |
| s | 38 | ve | 28 |

Figure 5. Confirmed 1-grams overview.

¹¹ According to this chart in Figure 4, about third of the sentences are translated better by *Neural-MT* (green area), third by *uedin-jhu-phrase* (red area) and for the last third the BLEU difference is negligible (or exactly zero). Also the confidence intervals in the bottom two charts are overlapping. Note that those two observations are not enough to conclude that the BLEU difference is not significant. For such claim we need a proper significance test, e.g. by paired bootstrap resampling, as shown in the upper right chart, which says that *Neural-MT* wins in only 36.7% of the resampled testsets, so it is worse than *uedin-jhu-phrase*, but not significantly (p-value=0.367 is higher than the conventional threshold 0.05, marked by the vertical red line at 95%).

4. Back-end

The back-end is responsible for monitoring a particular directory for new experiments and systems. The directory has to be specified in a configuration file. Consequently, a new experiment has to be imported in a new directory containing the source and the reference, and several sub-directories, one for each task (system). Additionally, metadata files allow adding names and descriptions to the imported experiments and tasks.

Once a new experiment directory or task sub-directory is detected, a set of background processes make sure that the sentences are analyzed and evaluated by the metrics and statistics. The sentences and the pre-computed results are entered into the database, so that they can be displayed without re-computation.

Sentences can be imported with conventional file moving options (`scp`, `sftp`). One additional possibility for fixed development cycles of wider development communities, is to sync the development translations with a storage cloud or a version management system, such as `git`, and include the commit ID in the description of each task. This has been tested in practice within the QTLeap project.¹²

5. System Description

5.1. Requirements and installation

MT-ComparEval has been designed and implemented to run in a Linux environment based on common free software utilities. The basic requirements needed to run MT-ComparEval are PHP 5.4 and SQLite 3.¹³ In the basic installation, no webserver is required to run MT-ComparEval because a simple webserver packaged with PHP is available. In a more optimized installation, webserver like Apache or Nginx can be also used.¹⁴ Database storage can be optimized through the use of more robust database engines, such as MySQL or MariaDB.

Concerning the installation, MT-ComparEval comes with a script `bin/install.sh` that installs locally all required PHP packages.

After the installation is finished, there are two scripts that are needed to run MT-ComparEval. First, `bin/server.sh`, which runs the application on `localhost:8080`. Second, `bin/watcher.sh`, which monitors the data directory data and imports all new experiments and tasks added to this folder.

¹²<http://qt leap.eu/>

¹³ <https://php.net/> and <https://www.sqlite.org/>

¹⁴ <http://httpd.apache.org/> and <https://www.nginx.com/>

5.2. Implementation

MT-ComparEval is an open-source tool developed in PHP, based on the Nette framework.¹⁵ This provides an architecture with a set of de-coupled and reusable PHP components, which is highly extensible. The visual interface is organized by a template engine, whereas there is easy access to the database and error logging. The modular interface can allow several extensions, such as the easy inclusion of additional automatic metrics.

The development process has been organized via a Git repository,¹⁶ so that it can continue as a community effort. All insights and ideas for extensions and improvements are collected as GitHub issues.

6. Summary and Further Work

We have outlined the main functionality of the evaluation panel and back-end of MT-ComparEval. The system includes several functions for aiding the process of evaluating systems with automatic scores, pairwise bootstrapping etc.

Although the evaluation interface already offers a vast amount of evaluation functions, we consider its expansion with more automatic metrics, which could possibly focus on specific issues and phenomena. Additionally, the possibility to have it run as a multi-user environment, where registered users can upload their systems' output and organize them in projects or based on their language pair, is considered to be a valuable extension. Finally, a set of exporting functions for tables and images in common flexible formats (comma-separated values, LaTeX, PDF) would be useful for aiding the authoring of academic papers and project reports.

Acknowledgments

This work was supported by the grants FP7-ICT-2013-10-610516 (QTLep), SVV 260 104, and it is using language resources hosted by the LINDAT/CLARIN project LM2010013 of the Ministry of Education, Youth and Sports.

Bibliography

- Aziz, Wilker, S Castilho, and Lucia Specia. PET: a Tool for Post-editing and Assessing Machine Translation. In *Eighth International Conference on Language Resources and Evaluation*, pages 3982–3987, Istanbul, Turkey, 2012. URL <http://wilkeraziz.github.io/dcs-site/publications/2012/AZIZ+LREC2012.pdf>.
- Berka, Jan, Ondřej Bojar, Mark Fishel, Maja Popović, and Daniel Zeman. Automatic MT Error Analysis: Hjerson Helping Addictor. In *Proceedings of the 8th International Conference*

¹⁵<http://nette.org>

¹⁶<https://github.com/choko/MT-ComparEval>

- on *Language Resources and Evaluation (LREC 2012)*, pages 2158–2163, İstanbul, Turkey, 2012. European Language Resources Association. ISBN 978-2-9517408-7-7.
- Bojar, Ondřej, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, USA, 2014. ACL. URL <http://www.aclweb.org/anthology/W/W14/W14-3302>.
- Bojar, Ondřej, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. Findings of the 2015 Workshop on Statistical Machine Translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46, Lisboa, Portugal, September 2015. Association for Computational Linguistics. URL <http://aclweb.org/anthology/W15-3001>.
- Callison-Burch, Chris, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. (Meta-) Evaluation of Machine Translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 136–158, Prague, Czech Republic, June 2007. ACL.
- Clark, Jonathan H., Chris Dyer, Alon Lavie, and Noah A. Smith. Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 176–181, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P11-2031>.
- Federmann, Christian. Appraise: An Open-Source Toolkit for Manual Phrase-Based Evaluation of Translations. In *LREC 2010*, pages 1731–1734, Valletta, Malta, May 2010. European Language Resources Association (ELRA). ISBN 2-9517408-6-7. URL http://www.lrec-conf.org/proceedings/lrec2010/pdf/197_Paper.pdf.
- Giménez, J and L Márquez. Asiya: An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation. *The Prague Bulletin of Mathematical Linguistics*, 2010. URL <http://ufal.mff.cuni.cz/pbml/94/art-gimenez-marques-evaluation.pdf>.
- Girardi, Christian, Luisa Bentivogli, Mohammad Amin Farajian, and Marcello Federico. MT-EQuAl: a Toolkit for Human Assessment of Machine Translation Output. In *COLING 2014*, pages 120–123, Dublin, Ireland, Aug. 2014. Dublin City University and ACL. URL <http://www.aclweb.org/anthology/C14-2026>.
- Koehn, Philipp. Statistical significance tests for machine translation evaluation. In Lin, Dekang and Dekai Wu, editors, *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain, 2004. ACL.
- Koehn, Philipp. An Experimental Management System. *The Prague Bulletin of Mathematical Linguistics*, 94:87–96, 2010. doi: 10.2478/v10108-010-0023-5. URL <http://ufal.mff.cuni.cz/pbml/94/art-koehn-ems.pdf>.
- Koehn, Philipp, Wade Shen, Marcello Federico, Nicola Bertoldi, Chris Callison-Burch, Brooke Cowan, Chris Dyer, Hieu Hoang, Ondrej Bojar, Richard Zens, Alexandra Constantin, Evan Herbst, and Christine Moran. Open Source Toolkit for Statistical Machine Translation. In *Proceedings of ACL*, pages 177–180, Prague, Czech Republic, June 2006.

- Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pages 311–318, Stroudsburg, PA, USA, 2002. ACL. URL <http://dx.doi.org/10.3115/1073083.1073135>.
- Popović, Maja. Hjerson: An Open Source Tool for Automatic Error Classification of Machine Translation Output. *The Prague Bulletin of Mathematical Linguistics*, 96:59–68, 2011. doi: 10.2478/v10108-011-0011-4. URL <http://ufal.mff.cuni.cz/pbml/96/art-popovic.pdf>.
- Riezler, Stefan and John T. Maxwell. On Some Pitfalls in Automatic Evaluation and Significance Testing for MT. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 57–64, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W/W05/W05-0908>.
- Zeman, Daniel, Mark Fishel, Jan Berka, and Ondřej Bojar. Addicter: What Is Wrong with My Translations? *The Prague Bulletin of Mathematical Linguistics*, 96:79–88, 2011. ISSN 0032-6585.

Address for correspondence:

Martin Popel

popel@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics

Faculty of Mathematics and Physics, Charles University in Prague

Malostranské náměstí 25, 118 00 Praha 1, Czech Republic



TmTriangulate: A Tool for Phrase Table Triangulation

Duc Tam Hoang, Ondřej Bojar

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

Abstract

Over the past years, pivoting methods, i.e. machine translation via a third language, gained respectable attention. Various experiments with different approaches and datasets have been carried out but the lack of open-source tools makes it difficult to replicate the results of these experiments. This paper presents a new tool for pivoting for phrase-based statistical machine translation by so called phrase-table triangulation. Besides the tool description, this paper discusses the strong and weak points of various triangulation techniques implemented in the tool.

1. Introduction

Training algorithms for statistical machine translation (SMT) generally rely on a large parallel corpus between the source language and target language. This paradigm may suffer from serious problems for under-resourced language pairs, for which such bilingual data are insufficient. In fact, if we randomly pick two living human languages, the pair will likely be under-resourced. Hence, most of the language pairs cannot benefit from standard SMT algorithms.

To alleviate the problem of data scarcity, *pivoting* has been introduced. It involves the use of another language, called *pivot language*, *bridge language* or the *third language*, to include resources available for the pivot language in the system. Over the years, a number of pivoting methods have been proposed, including system cascades, synthetic corpus, phrase table translation and most recently, phrase table triangulation.

Figure 1 shows a schematic overview of the SMT process and the interaction with various pivoting methods.

“System cascades” basically consist of translating the input from the source language into the pivot language, e.g. English, and then translating the obtained hypotheses into the target language. In the synthetic corpus method, the pivot side of a

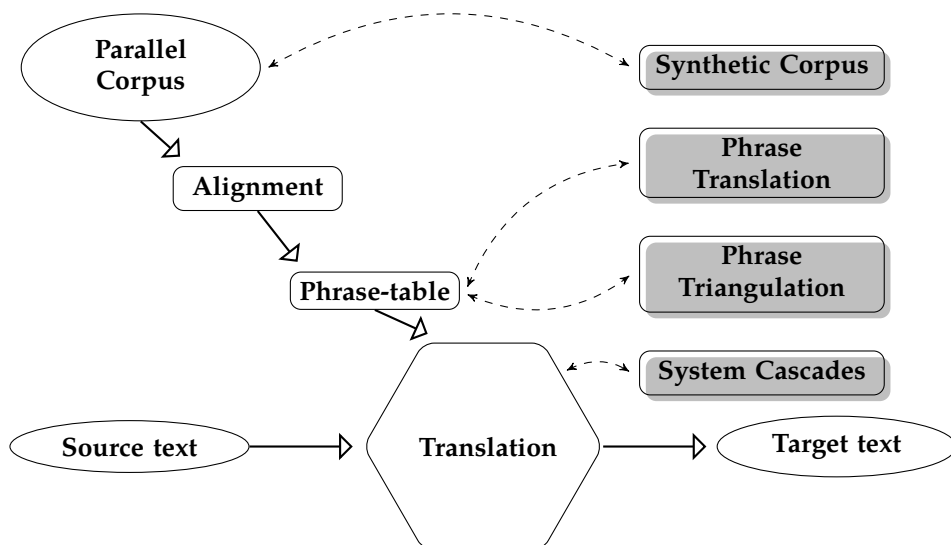


Figure 1: Pivoting methods

source-pivot or pivot-target parallel corpus is first translated to obtain a source-target corpus where one side is synthetic. A standard system is then trained from the obtained corpus. In the phrase table translation method, one side of an existing pivot-source or pivot-target phrase table is translated. And finally, the phrase table triangulation method (sometimes called simply *triangulation method*) combines two phrase tables, namely source-pivot and pivot-target, into an artificial source-target phrase table. Phrase table triangulation and translation thus manipulate directly with the internals of the SMT system, compared to system cascades, which uses source-to-pivot and pivot-to-target systems as black boxes, and synthetic corpus, which adjusts the training corpus.

Deploying system cascades in practice requires the two black-box systems running. *Phrase table triangulation* removes this requirement, capturing the new knowledge in a standard static file. This phrase table can then be used with any other SMT technique, see e.g. Zhu et al. (2014). One of the common operations is to merge several phrase tables for one language pair into one. The Moses toolkit (Koehn et al., 2007) includes a number of methods or tools for this: *alternative decoding paths*, *phrase table interpolation* (*tmcombine*; Sennrich, 2012) and *phrase table fill-up* (*combine-ptables*; Nakov, 2008; Bisazza et al., 2011).

In the past few years, promising results have been reported using phrase table triangulation methods (Cohn and Lapata, 2007; Razmara and Sarkar, 2013; Zhu et al.,

2014), but without releasing any open-source tool. We decided to fill this gap and implement an easy-to-use tool for phrase table triangulation in its several variants.

2. Phrase Table Triangulation

In short, phrase table triangulation fuses together source-pivot and pivot-target phrase tables, generating an artificial source-target phrase table as the output. Since each of the phrase tables usually consists of millions of phrase pairs, phrase table triangulation is computationally demanding (but lends itself relatively easily to parallelization).

When constructing the source-target table, we need to provide the set of:

1. source and target phrases, $\bar{s}-\bar{t}$,
2. word alignment a between them,¹
3. scores (direct and reverse phrase and lexical translation probabilities).

Two techniques were examined for the last step, namely *pivoting probabilities* (see Section 2.3; Cohn and Lapata, 2007; Utiyama and Isahara, 2007; Wu and Wang, 2007) and *pivoting co-occurrence counts* (see Section 2.4; Zhu et al., 2014).

2.1. Linking Source and Target Phrases

For source and target phrases, we do the most straightforward thing. We connect \bar{s} and \bar{t} whenever there exists a pivot phrase \bar{p} such that $\bar{s}-\bar{p}$ is listed in the source-pivot and $\bar{p}-\bar{t}$ is listed in the pivot-target phrase table. This approach however potentially springs serious problems.

Firstly, we do not check any context or meaning of the phrases, so an ambiguous pivot phrase \bar{p} can connect source and target phrases with totally unrelated meaning. This issue is more likely for short and frequent phrases.

Secondly, errors and omissions caused by noisy word alignments, which are unavoidable, are encountered twice. This leads to much higher level of noise in the final source-target table.

Thirdly, the noise boosts the number of common or short phrase pairs and omits a great proportion of large or rare phrase pairs. As the method relies on identical pivot phrases to link source phrases and target phrases, the longer the phrase is, the smaller the probability that there will be a pair.

And finally, we create the Cartesian product of the phrases, so the resulting phrase table is much larger than the size of the source phrase tables.

¹Strictly speaking, word alignment within phrases doesn't have to be provided, but the word alignment output of the final decoder run is useful in many applications. We also use the word alignment for pivoting lexical translation probabilities.

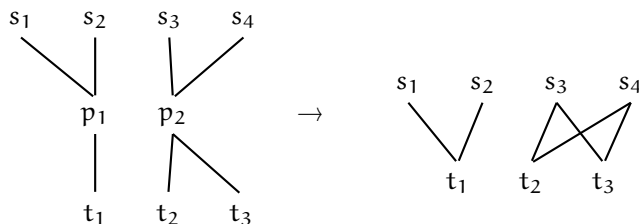


Figure 2: Constructing source-target alignment

2.2. Word Alignment for Linked Phrases

Given a triplet of source, pivot and target phrases $(\bar{s}, \bar{p}, \bar{t})$ and the source-pivot (a_{sp}) and pivot-target (a_{pt}) word alignments, we need to construct the source-target alignment a . We do this simply by tracing the alignments from each source word $s \in \bar{s}$ over any pivot word $p \in \bar{p}$ to each target word $t \in \bar{t}$ as illustrated in Figure 2. Formally:

$$(s, t) \in a \Leftrightarrow \exists p : (s, p) \in a_{sp} \ \& \ (p, t) \in a_{pt} \quad (1)$$

2.3. Pivoting Probabilities

Cohn and Lapata (2007) and Utiyama and Isahara (2007) considered triangulation as a generative probabilistic process which estimates new features based on features of the source-pivot and pivot-target phrase tables. This included an independence assumption of the conditional probabilities between \bar{s} , \bar{t} and \bar{p} :

$$\begin{aligned} p(\bar{s}|\bar{t}) &= \sum_{\bar{p}} p(\bar{s}|\bar{p}, \bar{t}) p(\bar{p}|\bar{t}) \\ &\approx \sum_{\bar{p}} p(\bar{s}|\bar{p}) p(\bar{p}|\bar{t}) \end{aligned} \quad (2)$$

Equation 2 finds the conditional over the pair (\bar{s}, \bar{t}) by going through all pivot phrases which are paired with both \bar{s} and \bar{t} . If we assume that each phrase \bar{p} represents a different sense, this can be viewed as including all phrase senses in the pivot language that \bar{s} and \bar{t} share.

The conditional probability can be simplified further by taking the maximum value instead of summing over all pivot phrases. This can potentially avoid the noise created

by alignment errors and corresponds to considering only the most prominent sense in the pivot language in our analogy. However, this may oversimplify the conditional probability and lead to information loss.

We apply the formula in Equation 2 to all four components of phrase pair scores: forward and direct phrase and lexically-weighted translation probabilities. Empirically, the resulting scores work reasonably well, but they are obviously not well defined probabilities.

2.4. Pivoting Co-Occurrence Counts

Zhu et al. (2014) introduced another approach to estimate the new features from the raw co-occurrence counts in the two source phrase tables.

Given the source-pivot co-occurrence count $c(\bar{s}, \bar{p})$ and the pivot-target count $c(\bar{p}, \bar{t})$, we need to select a function $f(\cdot, \cdot)$ that leads to a good estimate of the source-target count:

$$c(\bar{s}, \bar{t}) = \sum_p f(c(\bar{s}, \bar{p}), c(\bar{p}, \bar{t})) \tag{3}$$

There are four simple choices for $f(\cdot, \cdot)$ in Equation 3: the minimum, maximum, arithmetic mean and geometric mean. Zhu et al. (2014) considers the minimum as the best option.

Once the co-occurrence count for the phrase pair (\bar{s}, \bar{t}) in the synthetic source-target table is estimated, the direct and reverse phrase translation probabilities ϕ and their lexically-weighted variants p_w can be calculated using the standard procedure (Koehn et al., 2003). The reverse probabilities are calculated using the following formulas, the direct ones are estimated similarly:

$$\begin{aligned} \phi(\bar{s}|\bar{t}) &= \frac{c(\bar{s}, \bar{t})}{\sum_s c(\bar{s}, \bar{t})} \\ p_w(\bar{s}|\bar{t}, a) &= \prod_{i=1}^n \frac{1}{|j|(i, j) \in a|} \sum_{(i, j) \in a} w(s_i|t_j) \end{aligned} \tag{4}$$

In Equation 4, the lexical translation probability w between source word s and target word t must be computed beforehand as follows:

$$w(s|t) = \frac{c(s, t)}{\sum_{s'} c(s', t)} \tag{5}$$

Since we no longer have access to the *word* co-occurrence counts or lexical probabilities (the files `.f2e` and `.e2f` in Moses training), we estimate them from the pivoted

phrase table, i.e. the set of phrase pairs (\bar{s}, \bar{t}) that include the respective words s and t aligned:

$$c(s, t) = \sum_{\{(\bar{s}, \bar{t}) \mid s \in \bar{s} \& t \in \bar{t} \& (s, t) \in \alpha\}} c(\bar{s}, \bar{t}) \quad (6)$$

Pivoting co-occurrence counts is intuitively appealing because it leads to proper (maximum likelihood) probability estimates. On the other hand, it needs a good estimate for the co-occurrence counts in the first place. The approach works well if the parallel corpora are clean, of a similar size and distribution of words. Naturally, this is the case of multi-parallel corpora rather than two independent parallel corpora.

3. TmTriangulate

Our open-source tool for all the described variants of phrase table triangulation is designed to work with the Moses standard text format of phrase tables, making it compatible with other tools from the Moses toolkit, esp. tools for phrase table combination: *tmcombine* or *combine-ptables*.

As phrase table triangulation is a data-intensive operation, processing two huge files, it is not possible to keep the list of phrase pairs in memory. In fact, even the list of all phrase pairs associated with only one source phrase sometimes led to memory overload.

We therefore split triangulation into two steps: *triangulate* and *merge*. The first step, *triangulate*, is a *mergesort*-like process, handling phrase tables by travelling along the sorted pivot side of both input phrase tables. Once the same pivot phrase is spotted in both files, the source-target pair is established and emitted to a temporary output file with its (temporary) score values.

The second step sorts records of the temporary file and then merges values of all occurrences of the same source-target pair into one entry. Multi-threading is used in the second step for a better performance.

3.1. TmTriangulate Parameters

TmTriangulate command-line options are simple:

action select whether is it probabilities (`features_based`) or co-occurrence counts ("`counts_based`") that should be pivoted.

weight combination (-w) specifies handling for phrase pairs linked by more than one pivot phrase. The two accepted options `summation` and `maximization` correspond to summing over the pivot phrases or getting solely the maximum value for each score. If the value is not defined, `summation` option is chosen as default.

co-occurrence counts computation (-co) specifies the function f used to combine counts from the two input tables, see Section 2.4. Allowed values are: `min`, `max`, `a-mean` and `g-mean`.

| Parallel Corpus | Sentences | Tokens | | |
|--------------------|-----------|---------|---------|------------|
| | | Czech | English | Vietnamese |
| Czech-Vietnamese | 1.09M | 6.71M | – | 7.65M |
| Czech-English | 14.83M | 205.17M | 235.67M | – |
| English-Vietnamese | 1.35M | – | 12.78M | 12.49M |

Table 1: Sizes of parallel corpora used in our experiments.

mode (-m) clarifies the direction of component phrase tables, i.e. *source*→*pivot* or *pivot*→*source*. Accepted values are *pspt*, *sppt*, *pstp* and *sptp*, where the first pair of characters describes the source-pivot table and the second pair describes the pivot-target table.

source (-s) and target (-t) specify source-pivot and pivot-target phrase table files or directories with a given structure.

output phrase table (-o) and output lexical (-l) specify the output files. If the output file is not defined, *tmtriangulate* writes the source-target phrase table to the standard output.

For example, the following command constructs a Czech-Vietnamese phrase table by pivoting probabilities from the English-Czech (*en2cs.ttable.gz*) and English-Vietnamese (*en2vi.ttable.gz*) files:

```
./tmtriangulate.py features_based -m pspt \
-s en2cs.ttable.gz -t en2vi.ttable.gz
```

A detailed description of all parameters is provided along with the source code.

4. Experiments with Czech and Vietnamese

To illustrate the utility of *tmtriangulate*, we carry out an experiment with translation between Czech (*cs*) and Vietnamese (*vi*). English (*en*) is chosen as the sole pivot language.

4.1. Experiment Overview

The training data consist of three corpora: for *cs-en*, we use *CzEng 1.0* (Bojar et al., 2012) and for *cs-vi* and *en-vi*, we combine various sources including OPUS, TED talks and fragmented corpora published by previous works. Table 1 summarizes the sizes of our parallel data. Hence, the resources are unrelated and they are drastically different in size.

For completeness, our language model data are described in Table 2, we build standard 6-gram LMs with modified Kneser-Ney smoothing using KenLM (Heafield et al., 2013).

| Monolingual Corpus | Sentences | Tokens |
|--------------------|-----------|---------|
| Czech | 14.83M | 205.17M |
| Vietnamese | 1.81M | 48.98M |

Table 2: Sizes of monolingual corpora used for language models.

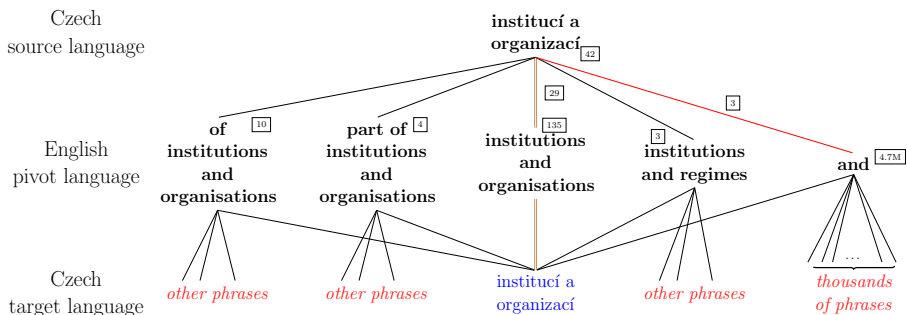


Figure 3: An example of triangulation with CzEng 1.0 corpus

Overall, the experiment is conducted with two directions: $cs \rightarrow vi$ and $vi \rightarrow cs$. We use *tmtriangulate* to combine phrase tables of $cs \rightarrow en$ and $en \rightarrow vi$ into the $cs \rightarrow vi$ and $vi \rightarrow cs$ tables. We use several settings for the triangulation to highlight the differences between them. Finally, we combine the best pivoted model with the standard phrase-based model extracted from an OPUS and TED direct parallel corpus between Czech and Vietnamese.

All systems are evaluated on a golden test set, obtained by manually translating the WMT13 test set² into Vietnamese, so there is no overlap between the training, tuning and evaluation data.

4.2. Noise Gained through Pivoting

We start with a quick manual inspection of the pivoted phrase tables. Differences in the domains and sizes of the source corpora are generally considered as the reasons behind the poor performance of triangulated models. Our analysis shows that alignment errors generate an immense amount of noise, degrading phrase table quality. For illustration purposes, we use the same phrase table twice, pivoting “from Czech to Czech” via English. This is actually one of the standard approaches to data-driven paraphrasing (Bannard and Callison-Burch, 2005) and obviously there cannot be any

²<http://www.statmt.org/wmt13/translation-task.html>

| Approach | Option | vi→cs BLEU | cs→vi BLEU |
|-------------------------------|-----------------|-------------|--------------|
| Pivoting probabilities | summation | 7.44 | 10.28 |
| Pivoting probabilities | maximization | 7.21 | 9.64 |
| Pivoting co-occurrence counts | minimum | 7.24 | 9.86 |
| Pivoting co-occurrence counts | maximum | 6.38 | 7.64 |
| Pivoting co-occurrence counts | arithmetic-mean | 6.25 | 6.95 |
| Pivoting co-occurrence counts | geometric-mean | 7.05 | 9.24 |
| Direct system | - | 7.62 | 10.59 |

Table 3: BLEU scores for phrase table triangulation for translation between Czech and Vietnamese via English.

discrepancies due to corpus size or domain. Yet, the pivoted phrase table contains many entries that distort the meaning. See Figure 3 for an example. The Czech phrase “institucí a organizací” by no doubt should be paired with a target phrase which has the sense: “institutions and organizations”. Indeed, the correct phrase pair has 29 co-occurrences, out of 135 appearances of “institutions and organizations” alone. The problem is that the single-word phrase “and” is listed as one of the possible translations and licenses a very large number of very distant phrases. It is just the 3 spurious co-occurrences with “and” that bring in the many bad phrases.

Our preliminary observations suggest that, after adding the pivot-target phrase table and estimating pivoted co-occurrence counts, the differences between good pairs and bad pairs get blurred. Estimating the new scores from source tables’ probabilities seems to keep the gap between good pairs and bad pairs wider. A more thorough analysis is nevertheless desirable.

4.3. Results of Pivoted Models Alone

Table 3 shows our first experimental results based on pivoted phrase tables.

The high level of noise leads to very large pivoted phrase tables with many bad phrases. The pivoted systems thus achieve relatively bad scores despite the large size of their phrase tables, many times larger than the size of the component phrase tables. Of the six triangulation options, the best one achieves results similar to the direct system, which is based on parallel cs-vi data.

The overall differences between the various triangulation approaches are not very big, especially concerning the high level of noise. We nevertheless see that for this set of languages and corpora, pivoting probabilities leads to better results than pivoting co-occurrence counts.

| Method | Table Size | vi→cs BLEU | cs→vi BLEU |
|---|------------|------------|------------|
| Direct System | 8.8M | 7.62 | 10.59 |
| Best Pivoted System | 61.5M | 7.44 | 10.28 |
| Linear Interpolation (<i>tmcombine</i>) | 69.3M | 8.33 | 11.98 |
| Alternative Decoding Paths | 8.8M/61.5M | 8.34 | 11.85 |

Table 4: Combining direct and pivoted phrase tables.

4.4. Combination with the Baseline Phrase Table

While the triangulation results did not improve over the baseline in the previous section, triangulation has reportedly brought gains *in combination* with the direct phrase table. Since the direct and the pivoted phrase tables have the same format, it is very easy to merge them.

We examine two options to combine the direct phrase table with the best pivoted phrase table: alternative decoding paths and phrase table interpolation. Alternative decoding paths in Moses use both tables at once and the standard MERT is used to optimize the (twice as big) set of weights, estimating the relative importance of the tables. Phrase table interpolation is implemented in *tmcombine* (among others) and merges the two tables with uniform weights before Moses is launched.

Table 4 confirms the reported results: the combined systems are significantly better than each of their components. We do not see much difference between alternative decoding paths and phrase table interpolation.

5. Conclusion

We discussed several options of pivoting, using a third language in machine translation. We focussed on phrase table triangulation and implemented a tool for several variants of the method. The tool, *tmtriangulate*, is freely available here:

<https://github.com/tamhd/MultiMT>

In our first experiment, phrase tables constructed by triangulation lead to results comparable but not better with the direct baseline translation. An improvement was achieved when we merged the direct and pivoted phrase tables with tools readily available in the Moses toolkit. It is however important to realize that different sets of languages, domains and corpora may show different behaviour patterns.

Acknowledgments

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreements n° 645452 (QT21) and

n° 644402 (HimL). The project was also supported by the grant SVV 260 104, and it is using language resources hosted by the LINDAT/CLARIN project LM2010013 of the Ministry of Education, Youth and Sports.

Bibliography

- Bannard, Colin and Chris Callison-Burch. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, ACL '05*, pages 597–604, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219914. URL <http://dx.doi.org/10.3115/1219840.1219914>.
- Bisazza, Arianna, Nick Ruiz, and Marcello Federico. Fill-up versus interpolation methods for phrase-based SMT adaptation. In *2011 International Workshop on Spoken Language Translation, IWSLT 2011, San Francisco, CA, USA, December 8-9, 2011*, pages 136–143, 2011. URL http://www.isca-speech.org/archive/iwslt_11/sltb_136.html.
- Bojar, Ondřej, Zdeněk Žabokrtský, Ondřej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jiří Maršík, Michal Novák, Martin Popel, and Aleš Tamchyna. The Joy of Parallelism with CzEng 1.0. In *Proceedings of LREC2012, Istanbul, Turkey, 2012*. ELRA, European Language Resources Association.
- Cohn, Trevor and Mirella Lapata. Machine Translation by Triangulation: Making Effective Use of Multi-Parallel Corpora. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic, 2007*. URL <http://aclweb.org/anthology-new/P/P07/P07-1092.pdf>.
- Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Sofia, Bulgaria, 8 2013*. URL http://khefield.com/professional/edinburgh/estimate_paper.pdf.
- Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical Phrase-Based Translation. In *HLT-NAACL, 2003*. URL <http://acl.ldc.upenn.edu/N/N03/N03-1017.pdf>.
- Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic, 2007*. URL <http://aclweb.org/anthology-new/P/P07/P07-2045.pdf>.
- Nakov, Preslav. Improving English-Spanish Statistical Machine Translation: Experiments in Domain Adaptation, Sentence Paraphrasing, Tokenization, and Recasing. In *Proceedings of the Third Workshop on Statistical Machine Translation, StatMT '08*, pages 147–150, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.
- Razmara, Majid and Anoop Sarkar. Ensemble Triangulation for Statistical Machine Translation. In *Sixth International Joint Conference on Natural Language Processing, IJCNLP 2013, Nagoya, Japan, October 14-18, 2013*, pages 252–260, 2013. URL <http://aclweb.org/anthology/I/I13/I13-1029.pdf>.

- Sennrich, Rico. Perplexity Minimization for Translation Model Domain Adaptation in Statistical Machine Translation. In *EACL 2012, 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France, April 23-27, 2012*, pages 539–549, 2012. URL <http://aclweb.org/anthology-new/E/E12/E12-1055.pdf>.
- Utiyama, Masao and Hitoshi Isahara. A Comparison of Pivot Methods for Phrase-Based Statistical Machine Translation. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA*, pages 484–491, 2007. URL <http://www.aclweb.org/anthology/N07-1061>.
- Wu, Hua and Haifeng Wang. Pivot Language Approach for Phrase-Based Statistical Machine Translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007. URL <http://aclweb.org/anthology-new/P/P07/P07-1108.pdf>.
- Zhu, Xiaoning, Zhongjun He, Hua Wu, Conghui Zhu, Haifeng Wang, and Tiejun Zhao. Improving Pivot-Based Statistical Machine Translation by Pivoting the Co-occurrence Count of Phrase Pairs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1665–1675, 2014. URL <http://aclweb.org/anthology/D/D14/D14-1174.pdf>.

Address for correspondence:

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



The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015

INSTRUCTIONS FOR AUTHORS

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

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

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

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

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

Authors receive a printed copy of the relevant issue of the PBML together with the original pdf files.

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