PBML

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

Joshua[1] 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`

Corresponding author: `post@cs.jhu.edu`
Cite as: Matt Post, Yuan Cao, Gaurav Kumar. Joshua 6: A phrase-based and hierarchical statistical machine translation system. The Prague Bulletin of Mathematical Linguistics No. 104, 2015, pp. 5–16.
doi: 10.1515/pralin-2015-0009.

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

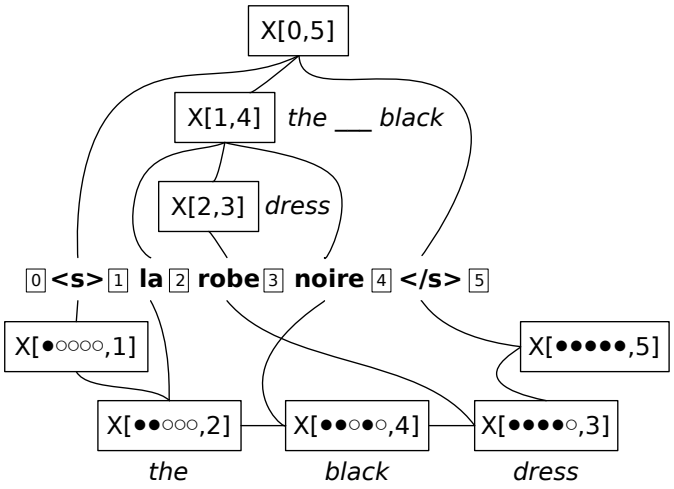


Figure 1. Shared hypergraph format when translating the French sentence la robe noire *into English with the hierarchical (above) and phrase-based (below) decoders. The nodes are states in the hypergraph and contain the nonterminal label (here, X) and the input span (hierarchical) or coverage vector 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:

[2]This 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 \
  --rundir 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.

[3] e.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.

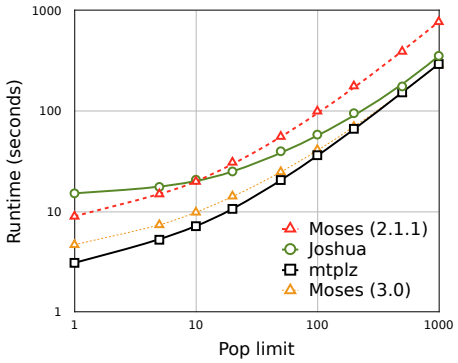
6. Experiments

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



	runtime (seconds)			
beam	Joshua	Moses 2	Moses 3	mtplz
1	15	9	5	3
5	18	15	7	5
10	21	20	10	7
20	25	31	14	11
50	40	56	25	21
100	58	100	41	36
200	95	178	70	67
500	176	393	151	154
1000	355	775	293	294

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 η=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 features are added, AdaGrad and PRO performed very well on the Spanish–English and

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.

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 learing 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

[4] Available 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.

[5] All 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