Joshua 5.0: Sparser, better, faster, server

Matt Post[1] and **Juri Ganitkevitch**[2] and **Luke Orland**[1] and **Jonathan Weese**[2] and **Yuan Cao**[2]
[1]Human Language Technology Center of Excellence
[2]Center for Language and Speech Processing
Johns Hopkins University

Chris Callison-Burch
Computer and Information Sciences Department
University of Pennsylvania

Abstract

We describe improvements made over the past year to Joshua, an open-source translation system for parsing-based machine translation. The main contributions this past year are significant improvements in both speed and usability of the grammar extraction and decoding steps. We have also rewritten the decoder to use a sparse feature representation, enabling training of large numbers of features with discriminative training methods.

1 Introduction

Joshua is an open-source toolkit[1] for hierarchical and syntax-based statistical machine translation of human languages with synchronous context-free grammars (SCFGs). The original version of Joshua (Li et al., 2009) was a port (from Python to Java) of the Hiero machine translation system introduced by Chiang (2007). It was later extended to support grammars with rich syntactic labels (Li et al., 2010). Subsequent efforts produced Thrax, the extensible Hadoop-based extraction tool for synchronous context-free grammars (Weese et al., 2011), later extended to support pivoting-based paraphrase extraction (Ganitkevitch et al., 2012). Joshua 5.0 continues our yearly update cycle.

The major components of Joshua 5.0 are:

§3.1 *Sparse features.* Joshua now supports an easily-extensible sparse feature implementation, along with tuning methods (PRO and kbMIRA) for efficiently setting the weights on large feature vectors.

§3.2 *Significant speed increases.* Joshua 5.0 is up to six times faster than Joshua 4.0, and also does well against hierarchical Moses, where end-to-end decoding (including model loading) of WMT test sets is as much as three times faster.

§3.3 *Thrax 2.0.* Our reengineered Hadoop-based grammar extractor, Thrax, is up to 300% faster while using significantly less intermediate disk space.

§3.4 *Many other features.* Joshua now includes a server mode with fair round-robin scheduling among and within requests, a bundler for distributing trained models, improvements to the Joshua pipeline (for managing end-to-end experiments), and better documentation.

2 Overview

Joshua is an end-to-end statistical machine translation toolkit. In addition to the decoder component (which performs the actual translation), it includes the infrastructure needed to prepare and align training data, build translation and language models, and tune and evaluate them.

This section provides a brief overview of the contents and abilities of this toolkit. More information can be found in the online documentation (`joshua-decoder.org/5.0/`).

2.1 The Pipeline: Gluing it all together

The Joshua pipeline ties together all the infrastructure needed to train and evaluate machine translation systems for research or industrial purposes. Once data has been segmented into parallel training, development, and test sets, a single invocation of the pipeline script is enough to invoke this entire infrastructure from beginning to end. Each step is

[1] `joshua-decoder.org`

Proceedings of the Eighth Workshop on Statistical Machine Translation, pages 206–212,
Sofia, Bulgaria, August 8-9, 2013 ©2013 Association for Computational Linguistics

broken down into smaller steps (e.g., tokenizing a file) whose dependencies are cached with SHA1 sums. This allows a reinvoked pipeline to reliably skip earlier steps that do not need to be recomputed, solving a common headache in the research and development cycle.

The Joshua pipeline is similar to other "experiment management systems" such as Moses' Experiment Management System (EMS), a much more general, highly-customizable tool that allows the specification and parallel execution of steps in arbitrary acyclic dependency graphs (much like the UNIX `make` tool, but written with machine translation in mind). Joshua's pipeline is more limited in that the basic pipeline skeleton is hard-coded, but reduced versatility covers many standard use cases and is arguably easier to use.

The pipeline is parameterized in many ways, and all the options below are selectable with command-line switches. Pipeline documentation is available online.

2.2 Data preparation, alignment, and model building

Data preparation involves data normalization (e.g., collapsing certain punctuation symbols) and tokenization (with the Penn treebank or user-specified tokenizer). Alignment with GIZA++ (Och and Ney, 2000) and the Berkeley aligner (Liang et al., 2006b) are supported.

Joshua's builtin grammar extractor, Thrax, is a Hadoop-based extraction implementation that scales easily to large datasets (Ganitkevitch et al., 2013). It supports extraction of both Hiero (Chiang, 2005) and SAMT grammars (Zollmann and Venugopal, 2006) with extraction heuristics easily specified via a flexible configuration file. The pipeline also supports GHKM grammar extraction (Galley et al., 2006) using the extractors available from Michel Galley[2] or Moses.

SAMT and GHKM grammar extraction require a parse tree, which are produced using the Berkeley parser (Petrov et al., 2006), or can be done outside the pipeline and supplied as an argument.

2.3 Decoding

The Joshua decoder is an implementation of the CKY+ algorithm (Chappelier et al., 1998), which generalizes CKY by removing the requirement that the grammar first be converted to Chomsky Normal Form, thereby avoiding the complexities of explicit binarization schemes (Zhang et al., 2006; DeNero et al., 2009). CKY+ maintains cubic-time parsing complexity (in the sentence length) with Earley-style implicit binarization of rules. Joshua permits arbitrary SCFGs, imposing no limitation on the rank or form of grammar rules.

Parsing complexity is still exponential in the scope of the grammar,[3] so grammar filtering remains important. The default Thrax settings extract only grammars with rank 2, and the pipeline implements scope-3 filtering (Hopkins and Langmead, 2010) when filtering grammars to test sets (for GHKM).

Joshua uses cube pruning (Chiang, 2007) with a default pop limit of 100 to efficiently explore the search space. Other decoder options are too numerous to mention here, but are documented online.

2.4 Tuning and testing

The pipeline allows the specification (and optional linear interpolation) of an arbitrary number of language models. In addition, it builds an interpolated Kneser-Ney language model on the target side of the training data using KenLM (Heafield, 2011; Heafield et al., 2013), BerkeleyLM (Pauls and Klein, 2011) or SRILM (Stolcke, 2002).

Joshua ships with MERT (Och, 2003) and PRO implementations. Tuning with k-best batch MIRA (Cherry and Foster, 2012) is also supported via callouts to Moses.

3 What's New in Joshua 5.0

3.1 Sparse features

Until a few years ago, machine translation systems were for the most part limited in the number of features they could employ, since the line-based optimization method, MERT (Och, 2003), was not able to efficiently search over more than tens of feature weights. The introduction of discriminative tuning methods for machine translation (Liang et al., 2006a; Tillmann and Zhang, 2006; Chiang et al., 2008; Hopkins and May, 2011) has made it possible to tune large numbers of features in statistical machine translation systems, and open-

[2] `nlp.stanford.edu/~mgalley/software/stanford-ghkm-latest.tar.gz`

[3] Roughly, the number of consecutive nonterminals in a rule (Hopkins and Langmead, 2010).

source implementations such as Cherry and Foster (2012) have made it easy.

Joshua 5.0 has moved to a sparse feature representation internally. First, to clarify terminology, a feature as implemented in the decoder is actually a template that can introduce any number of actual features (in the standard machine learning sense). We will use the term *feature function* for these templates and *feature* for the individual, traditional features that are induced by these templates. For example, the (typically dense) features stored with the grammar on disk are each separate features contributed by the PHRASEMODEL feature function template. The LANGUAGEMODEL template contributes a single feature value for each language model that was loaded.

For efficiency, Joshua does not store the entire feature vector during decoding. Instead, hypergraph nodes maintain only the best cumulative score of each incoming hyperedge, and the edges themselves retain only the hyperedge delta (the inner product of the weight vector and features incurred by that edge). After decoding, the feature vector for each edge can be recomputed and explicitly represented if that information is required by the decoder (for example, during tuning).

This functionality is implemented via the following feature function interface, presented here in simplified pseudocode:

```
interface FeatureFunction:
  apply(context, accumulator)
```

The `context` comprises fixed pieces of the input sentence and hypergraph:

- the hypergraph edge (which represents the SCFG rule and sequence of tail nodes)
- the complete source sentence
- the input span

The `accumulator` object's job is to accumulate feature (name,value) pairs fired by a feature function during the application of a rule, via another interface:

```
interface Accumulator:
  add(feature_name, value)
```

The accumulator generalization[4] permits the use of a single feature-gathering function for two accumulator objects: the first, used during decoding, maintains only a weighted sum, and the second, used (if needed) during k-best extraction, holds onto the entire sparse feature vector.

For tuning large sets of features, Joshua supports both PRO (Hopkins and May, 2011), an in-house version introduced with Joshua 4.0, and k-best batch MIRA (Cherry and Foster, 2012), implemented via calls to code provided by Moses.

3.2 Performance improvements

We introduced many performance improvements, replacing code designed to get the job done under research timeline constraints with more efficient alternatives, including smarter handling of locking among threads, more efficient (non string-based) computation of dynamic programming state, and replacement of fixed class-based array structures with fixed-size literals.

We used the following experimental setup to compare Joshua 4.0 and 5.0: We extracted a large German-English grammar from all sentences with no more than 50 words per side from Europarl v.7 (Koehn, 2005), News Commentary, and the Common Crawl corpora using Thrax default settings. After filtering against our test set (newstest2012), this grammar contained 70 million rules. We then trained three language models on (1) the target side of our grammar training data, (2) English Gigaword, and (3) the monolingual English data released for WMT13. We tuned a system using kbMIRA and decoded using KenLM (Heafield, 2011). Decoding was performed on 64-core 2.1 GHz AMD Opteron processors with 256 GB of available memory.

Figure 1 plots the end-to-end runtime[5] as a function of the number of threads. Each point in the graph is the minimum of at least fifteen runs computed at different times over a period of a few days. The main point of comparison, between Joshua 4.0 and 5.0, shows that the current version is up to 500% faster than it was last year, especially in multithreaded situations.

For further comparison, we took these models, converted them to hierarchical Moses format, and then decoded with the latest version.[6] We compiled Moses with the recommended optimization settings[7] and used the in-memory (SCFG) gram-

[4] Due to Kenneth Heafield.

[5] i.e., including model loading time and grammar sorting

[6] The latest version available on Github as of June 7, 2013

[7] With tcmalloc and the following compile flags: `--max-factors=1 --kenlm-max-order=5 debug-symbols=off`

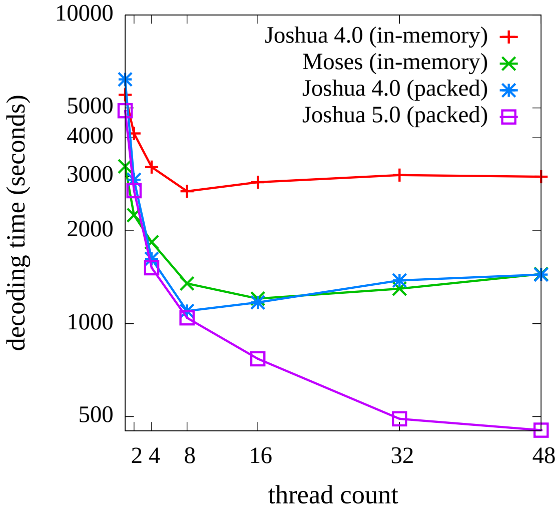

Figure 1: End-to-end runtime as a function of the number of threads. Each data point is the minimum of at least fifteen different runs.

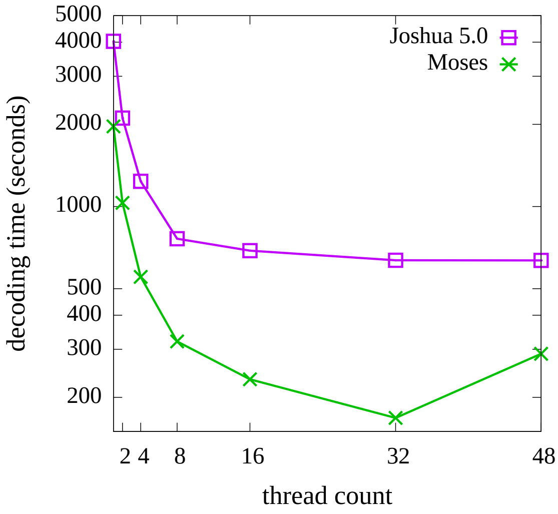

Figure 2: Decoding time alone.

mar format. BLEU scores were similar.[8] In this end-to-end setting, Joshua is about 200% faster than Moses at high thread counts (Figure 1).

Figure 2 furthers the Moses and Joshua comparison by plotting only decoding time (subtracting out model loading and sorting times). Moses' decoding speed is 2–3 times faster than Joshua's, suggesting that the end-to-end gains in Figure 1 are due to more efficient grammar loading.

3.3 Thrax 2.0

The Thrax module of our toolkit has undergone a similar overhaul. The rule extraction code was rewritten to be easier to understand and extend, allowing, for instance, for easy inclusion of alternative nonterminal labeling strategies.

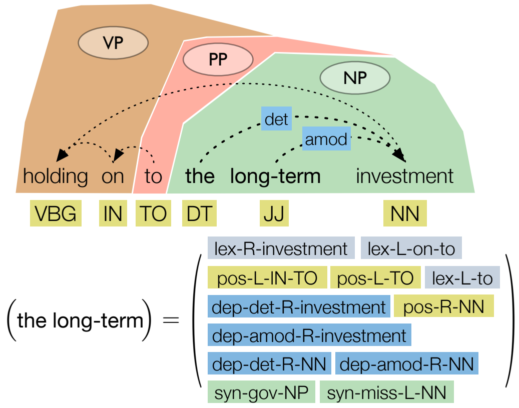

Figure 3: Here, position-aware lexical and part-of-speech n-gram features, labeled dependency links, and features reflecting the phrase's CCG-style label NP/NN are included in the context vector.

We optimized the data representation used for the underlying map-reduce framework towards greater compactness and speed, resulting in a 300% increase in extraction speed and an equivalent reduction in disk I/O (Table 1). These gains enable us to extract a syntactically labeled German-English SAMT-style translation grammar from a bitext of over 4 million sentence pairs in just over three hours. Furthermore, Thrax 2.0 is capable of scaling to very large data sets, like the composite bitext used in the extraction of the paraphrase collection PPDB (Ganitkevitch et al., 2013), which counted 100 million sentence pairs and over 2 billion words on the English side.

Furthermore, Thrax 2.0 contains a module focused on the extraction of compact distributional signatures over large datasets. This *distributional* mode collects contextual features for n-gram phrases, such as words occurring in a window around the phrase, as well as dependency-based and syntactic features. Figure 3 illustrates the feature space. We then compute a bit signature from the resulting feature vector via a randomized locality-sensitive hashing projection. This yields a compact representation of a phrase's typical context. To perform this projection Thrax relies on the Jerboa toolkit (Van Durme, 2012). As part of the PPDB effort, Thrax has been used to extract rich distributional signatures for 175 million 1-to-4-gram phrases from the Annotated Gigaword corpus (Napoles et al., 2012), a parsed and pro-

[8] 22.88 (Moses), 22.99 (Joshua 4), and 23.23 (Joshua 5).

Rules	Cs-En 112M		Fr-En 357M		De-En 202M		Es-En 380M	
	Space	Time	Space	Time	Space	Time	Space	Time
Joshua 4.0	120GB	112 min	364GB	369 min	211GB	203 min	413GB	397 min
Joshua 5.0	31GB	25 min	101GB	81 min	56GB	44 min	108GB	84 min
Difference	-74.1%	-77.7%	-72.3%	-78.0%	-73.5%	-78.3%	-73.8%	-78.8%

Table 1: Comparing Hadoop's intermediate disk space use and extraction time on a selection of Europarl v.7 Hiero grammar extractions. Disk space was measured at its maximum, at the input of Thrax's final grammar aggregation stage. Runtime was measured on our Hadoop cluster with a capacity of 52 mappers and 26 reducers. On average Thrax 2.0, bundled with Joshua 5.0, is up to 300% faster and more compact.

cessed version of the English Gigaword (Graff et al., 2003).

Thrax is distributed with Joshua and is also available as a separate download.[9]

3.4 Other features

Joshua 5.0 also includes many features designed to increase its usability. These include:

- A TCP/IP server architecture, designed to handle multiple sets of translation requests while ensuring fairness in thread assignment both across and within these connections.
- Intelligent selection of translation and language model training data using cross-entropy difference to rank training candidates (Moore and Lewis, 2010; Axelrod et al., 2011) (described in detail in Orland (2013)).
- A bundler for easy packaging of trained models with all of its dependencies.
- A year's worth of improvements to the Joshua pipeline, including many new features and supported options, and increased robustness to error.
- Extended documentation.

4 WMT Submissions

We submitted a constrained entry for all tracks except English-Czech (nine in total). Our systems were constructed in a straightforward fashion and without any language-specific adaptations using the Joshua pipeline. For each language pair, we trained a Hiero system on all sentences with no more than fifty words per side in the Europarl, News Commentary, and Common Crawl corpora. We built two interpolated Kneser-Ney language models: one from the monolingual News Crawl corpora (2007–2012), and another from the target side of the training data. For systems translating into English, we added a third language model built on Gigaword. Language models were combined linearly into a single language model using interpolation weights from the tuning data (newstest2011). We tuned our systems with kbMIRA. For truecasing, we used a monolingual translation system built on the training data, and finally detokenized with simple heuristics.

5 Summary

The 5.0 release of Joshua is the result of a significant year-long research, engineering, and usability effort that we hope will be of service to the research community. User-friendly packages of Joshua are available from `joshua-decoder.org`, while developers are encouraged to participate via `github.com/joshua-decoder/joshua`. Mailing lists, linked from the main Joshua page, are available for both.

Acknowledgments Joshua's sparse feature representation owes much to discussions with Colin Cherry, Barry Haddow, Chris Dyer, and Kenneth Heafield at MT Marathon 2012 in Edinburgh.

This material is based on research sponsored by the NSF under grant IIS-1249516 and DARPA under agreement number FA8750-13-2-0017 (the DEFT program). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained in this publication are those of the authors and should not be interpreted as representing official policies or endorsements of DARPA or the U.S. Government.

[9] `github.com/joshua-decoder/thrax`

References

Amittai Axelrod, Xiaodong He, and Jianfeng Gao. 2011. Domain adaptation via pseudo in-domain data selection. In *Proceedings of EMNLP*, pages 355–362, Edinburgh, Scotland, UK., July.

J.C. Chappelier, M. Rajman, et al. 1998. A generalized CYK algorithm for parsing stochastic CFG. In *First Workshop on Tabulation in Parsing and Deduction (TAPD98)*, pages 133–137.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proceedings of NAACL-HLT*, pages 427–436, Montréal, Canada, June.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proceedings of EMNLP*, Waikiki, Hawaii, USA, October.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of ACL*, Ann Arbor, Michigan.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

John DeNero, Adam Pauls, and Dan Klein. 2009. Asynchronous binarization for synchronous grammars. In *Proceedings of ACL*, Suntec, Singapore, August.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of ACL/COLING*, Sydney, Australia, July.

Juri Ganitkevitch, Yuan Cao, Jonathan Weese, Matt Post, and Chris Callison-Burch. 2012. Joshua 4.0: Packing, PRO, and paraphrases. In *Proceedings of the Workshop on Statistical Machine Translation*.

Juri Ganitkevitch, Chris Callison-Burch, and Benjamin Van Durme. 2013. Ppdb: The paraphrase database. In *Proceedings of HLT/NAACL*.

D. Graff, J. Kong, K. Chen, and K. Maeda. 2003. English gigaword. *Linguistic Data Consortium, Philadelphia*.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Proceedings of ACL*, Sofia, Bulgaria, August.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 187–197. Association for Computational Linguistics.

Mark Hopkins and Greg Langmead. 2010. SCFG decoding without binarization. In *Proceedings of EMNLP*, pages 646–655.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of EMNLP*.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. 2009. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Workshop on Statistical Machine Translation*, Athens, Greece, March.

Zhifei Li, 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. 2010. Joshua 2.0: a toolkit for parsing-based machine translation with syntax, semirings, discriminative training and other goodies. In *Proceedings of the Workshop on Statistical Machine Translation*.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006a. An end-to-end discriminative approach to machine translation. In *Proceedings of ACL/COLING*.

Percy Liang, Ben Taskar, and Dan Klein. 2006b. Alignment by agreement. In *Proceedings of NAACL*, pages 104–111, New York City, USA, June.

Robert C. Moore and William Lewis. 2010. Intelligent selection of language model training data. In *Proceedings of ACL (short papers)*, pages 220–224.

Courtney Napoles, Matt Gormley, and Benjamin Van Durme. 2012. Annotated gigaword. In *Proceedings of AKBC-WEKEX 2012*.

F. J. Och and H. Ney. 2000. Improved statistical alignment models. In *Proceedings of ACL*, pages 440–447, Hong Kong, China, October.

Franz Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, Sapporo, Japan.

Luke Orland. 2013. Intelligent selection of translation model training data for machine translation with TAUS domain data: A summary. Master's thesis, Johns Hopkins University, Baltimore, Maryland, June.

Adam Pauls and Dan Klein. 2011. Faster and smaller n-gram language models. In *Proceedings of ACL*, pages 258–267, Portland, Oregon, USA, June.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of ACL*, Sydney, Australia, July.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*.

Christoph Tillmann and Tong Zhang. 2006. A discriminative global training algorithm for statistical mt. In *Proceedings of ACL/COLING*, pages 721–728, Sydney, Australia, July.

Benjamin Van Durme. 2012. Jerboa: A toolkit for randomized and streaming algorithms. Technical Report 7, Human Language Technology Center of Excellence, Johns Hopkins University.

Jonathan Weese, Juri Ganitkevitch, Chris Callison-Burch, Matt Post, and Adam Lopez. 2011. Joshua 3.0: Syntax-based machine translation with the Thrax grammar extractor. In *Proceedings of the Workshop on Statistical Machine Translation*.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of HLT/NAACL*.

Andreas Zollmann and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, New York, New York.