PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 47–56

Decoding in Joshua

Open Source, Parsing-Based Machine Translation

Zhifei Li, Chris Callison-Burch, Sanjeev Khudanpur, Wren Thornton

Abstract

We describe a scalable decoder for parsing-based machine translation. The decoder is written in Java and implements all the essential algorithms described in (Chiang, 2007) and (Li and Khudanpur, 2008b): chart-parsing, n-gram language model integration, beam- and cube-pruning, and k-best extraction. Additionally, parallel and distributed computing techniques are exploited to make it scalable. We demonstrate experimentally that our decoder is more than 30 times faster than a baseline decoder written in Python.

1. Motivation

Large-scale parsing-based statistical machine translation has made significant progress in the last few years. The systems being developed differ in whether they use source- or target-language syntax. For instance, the hierarchical translation system of Chiang (2007) extracts a synchronous grammar from pairs of strings, whereas Quirk, Menezes, and Cherry (2005), Liu, Liu, and Lin (2006) and Huang, Knight, and Joshi (2006) perform syntactic analyses in the source-language, and Galley et al. (2006) uses target-language syntax.

A critical component in parsing-based MT systems is the decoder, which is complex to implement and scale up for large data sets. Most of the systems described above employ tailor-made, dedicated decoders that are not open-source, which results in a high barrier to entry for other researchers in the field. However, with the algorithms proposed in (Huang and Chiang, 2005, Chiang, 2007, Huang and Chiang, 2007), it is possible to develop a general-purpose decoder that can be used by all the parsing-based systems. In this paper, we describe an important first-step towards an extensible, general-purpose, scalable, and open-source parsing-based MT decoder. Our decoder is written in Java and implements all the essential algorithms described in (Chiang, 2007): chart-parsing, n-gram language model integration, beam- and

Please cite this article as: Zhifei Li, Chris Callison-Burch, Sanjeev Khudanpur, Wren Thornton, Decoding in Joshua: Open Source, Parsing-Based Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 47–56.

cube-pruning, and unique k-best extraction. Additionally, parallel and distributed computing techniques are exploited to make it scalable.

We demonstrate experimentally that our decoder is 38 times faster than a previous decoder written in Python. Furthermore, the distributed computing permits improving translation quality via large-scale language models. The decoder has been used to translate roughly a million sentences in a parallel corpus for large-scale discriminative training experiments (Li and Khudanpur, 2008a). The decoder has also been successfully used by other researchers. For example, (Chen et al., 2008) have demonstrated that our decoder achieves performance competitive with Moses (Koehn et al., 2007), another major open-source machine translation toolkit. We hope the release of the decoder will greatly contribute the progress of the syntax-based machine translation research.

2. Parsing-based MT Decoder

In this section, we discuss the core algorithms implemented in our decoder. These algorithms have been discussed by (Chiang, 2007) in detail, and we recapitulate the essential parts here for completeness.[1]

2.1. Grammar Formalism

Our decoder assumes a probabilistic synchronous context-free grammar (SCFG). Following the notation in (Venugopal, Zollmann, and Vogel, 2007), a probabilistic SCFG comprises a set of source-language terminal symbols, T_S, a set of target-language terminal symbols, T_T, a shared set of nonterminal symbols, N, and a set of rules of the form

$$X \rightarrow \langle \gamma, \alpha, \sim, w \rangle \tag{1}$$

where $X \in N$, $\gamma \in [N \cup T_S]^*$ is a (mixed) sequence of nonterminals and source terminals, $\alpha \in [N \cup T_T]^*$ is a sequence of nonterminals and target terminals, $\sim$ is a one-to-one correspondence or *alignment* between the nonterminal elements of γ and α, and $w \geq 0$ is a weight assigned to the rule. An illustrative rule for Chinese-to-English translation is

$$\mathrm{NP} \rightarrow \langle\, \mathrm{NP}_0 \text{ 的 } \mathrm{NP}_1 \,,\, \mathrm{NP}_1 \textit{ of } \mathrm{NP}_0 \,\rangle$$

where the Chinese word 的 (pronounced *de* or *di*) means *of*, and the alignment, encoded via subscripts on the nonterminals, causes the two noun phrases around 的 to be reordered around *of* in the translation. The rule weight is omitted in this example.

A bilingual SCFG derivation is analogous to a monolingual CFG derivation. It begins with a pair of *aligned* start symbols. At each step, an *aligned* pair of nonterminals is rewritten as the two corresponding components of a single rule. In this sense, the derivations are generated synchronously.

[1] Most of the descriptions here are adopted from (Li and Khudanpur, 2008b).

Our decoder presently handles SCFGs of the kind extracted by Heiro (Chiang, 2007), but is easily extensible to more general SCFGs and closely related formalisms such as synchronous tree substitution grammars (Eisner, 2003, Chiang, 2006).

2.2. MT Decoding as Chart Parsing

Given a source-language sentence, f^*, the decoder must find the target-language yield, $e(D)$, of the derivation D which has the best composite weight, $w(D)$, among all derivations whose source-language yield, $f(D)$, is the source-language sentence. Or equationally,

$$e^* = e\left(\underset{D\,:\,f(D)=f^*}{\arg\max}\; w(D)\right) \tag{2}$$

The composite weight is a linear combination of feature function weights and feature function values. General feature functions include translation model features, language model features, and word penalty features.

The actual decoding algorithm maintains a *chart*, which contains an array of *cells*. Each cell in turn maintains a list of proven *items*. The parsing process starts with the axioms, and proceeds by applying the inference rules repeatedly to prove new items until proving a goal item. Whenever the parser proves a new item, it adds the item to the appropriate chart cell. The new item also maintains backpointers to antecedent items, which are used for k-best extraction, as discussed in Section 2.4 below.

In a SCFG-based decoder, an *item* is identified by its source-language span, left-side nonterminal label, and left- and right-contexts for the target-language n-gram LM. Therefore, in a given cell, the maximum possible number of items is $O(|N|\,|T_T|^{2(n-1)})$, and the worst case decoding complexity is

$$O\left(l^3\,|N|^K\,|T_T|^{2K(n-1)}\right) \tag{3}$$

where K is the maximum number of nonterminal pairs per rule and l is the source-language sentence length (Venugopal, Zollmann, and Vogel, 2007).

2.3. Pruning in a Decoder

Severe pruning is needed in order to make the decoding computationally feasible for SCFGs with large target-language vocabularies and detailed nonterminal sets. In our decoder, we incorporate two pruning techniques described by (Chiang, 2007, Huang and Chiang, 2007). For *beam pruning*, in each cell, we discard all items whose weight is β-times worse than the weight of the best item in the same cell. If too many items pass that relative threshold, then only the top b items by weight are retained in each cell. When applying an inference rule to combine smaller items and obtain a larger item, we use *cube pruning* to simulate k-best extraction in each destination cell, discarding combinations which lead to an item whose weight is worse than the best item in that cell by a margin of ϵ.

2.4. Hyper-graphs and k-best Extraction

For each source-language sentence the output of the chart-parsing algorithm may be treated as a *hyper-graph* representing a set of likely derivation hypotheses. Briefly, a hyper-graph is a set of *vertices* and *hyper-edge*s, with each hyper-edge connecting a *set* of antecedent vertices to a consequent vertex, and a special vertex designated as the *target vertex*. In parsing parlance, a vertex corresponds to an item in the chart, a hyper-edge corresponds to a SCFG rule with the nonterminals on the right-side replaced by back-pointers to antecedent items, and the target vertex corresponds to the goal item[2].

Given a hyper-graph for a source-language sentence f^*, we use the k-best extraction algorithm of (Huang and Chiang, 2005) to extract its k most likely translations. Moreover, since many different derivations may lead to the same target-language yield $e(D)$, we adopt the modification described in (Huang, Knight, and Joshi, 2006) to efficiently generate the *unique* k best translations of f^*.

3. Underlying Methodologies

When designing our decoder we applied principles of software engineering to improve usability and hence utility to open-source users. Our three major design goals are: extendibility, end-to-end coherence, and scalability.

3.1. Extendibility

To make Joshua a suitable baseline for future research it is necessary that it be easily extended by other researchers. As befitting a project of its size, the Joshua code is organized into separate *packages* for each major aspect of functionality (e.g. chart parsing, feature functions, and hyper-graph algorithms). In this way it is clear which files contribute to a given functionality and researchers can focus on a single package without worrying about the rest of the system.

Illicit interactions and unseen dependencies are a common hinderance to extensibility in large projects. To minimize these problems, all extensible components are defined by Java *interfaces*. The interfaces are designed to be minimalistic so that they do not hinder radical departures from current implementations, such as using per-sentence or non-trie-based translation grammars. Where there is a clear point of departure for research, a basic implementation of each interface is provided as an *abstract class* to minimize the work necessary for new extensions.

A non-exhaustive list of future extensions we envisioned when designing our interfaces include:

- Using a new decoding algorithm such as agenda-based parsing, instead of the default CKY algorithm;

[2]In a decoder integrating an n-gram LM, there may be multiple goal items due to different LM contexts. However, one can image a *single* goal item identified by the span $[0, n]$ and the goal nonterminal S, but not by the LM contexts.

- Adding new pruning algorithms, beside the already implemented beam- and cube-pruning;
- Using grammars with linguistic syntax such as the grammar described in (Galley et al., 2006, Venugopal and Zollmann, 2009), rather than Hiero-style grammars;
- Handling non-SCFG grammar formalisms, e.g. synchronous tree substitution grammars (Eisner, 2003);
- Adding new feature functions, e.g. the source-side syntax constraints described by (Marton and Resnik, 2008);
- Using novel language models like the bloom-filter LM described in (Talbot and Osborne, 2007), not just ARPA backoff n-gram models;
- Adding new algorithms that operate on the hyper-graph, for example, hyper-graph reranking or discriminative training over the hyper-graph.

3.2. End-To-End Cohesion

There are many components to a machine translation pipeline aside from the decoder. One of the great difficulties with current MT pipelines is that these diverse components are often designed by separate groups and have different file format and interaction requirements. This leads to a large investment in scripts to convert formats and connect the different components, and often leads to untenable and non-portable projects as well as hindering repeatability of experiments.

To combat these issues, the Joshua toolkit integrates other critical components of the machine translation pipeline as well as the decoder. Two critical components being integrated are suffix-array grammar extraction (Callison-Burch, Bannard, and Schroeder, 2005, Lopez, 2007) and minimum error rate training (MERT) (Och, 2003, Bertoldi, Haddow, and Fouet, 2009, Zaidan, 2009). Additional components we hope to integrate include tools for building language models and generating word alignments, as well as a general infrastructure for configuring and connecting segments of the pipeline.

For researchers who have already invested much work into their pipelines, the decoder can be treated as a stand-alone tool and does not rely on the rest of the toolkit we provide.

3.3. Scalability

Our third design goal was to ensure that the decoder is scalable to large models and data sets. The parsing and pruning algorithms are carefully implemented with dynamic programming strategies, and efficient data structures are used to minimize overhead.

The integration of suffix-array grammar extraction and MERT also contributes to scalability. Suffix arrays are compact data structures which can store many more n-grams than a traditional phrase table with the same memory footprint. They are also amenable to extracting small per-sentence grammars on the fly, rather than needing a monolithic grammar for the entire test set. With MERT integration we do not need to start a new decoder instance each iteration, which means we can load the grammar into memory once (an expensive task compared to the decoding time itself) instead of repeatedly.

We also implement *parallel decoding* and a *distributed language model*. Parallel decoding is able to exploit multi-core and multi-processor architectures by translating multiple sentences in separate threads and storing the language model and translation grammar in shared memory. Enabling the distributed language model reduces memory pressure and makes it feasible to use large LMs by running the LM on a separate machine from the decoder or decoders. More details on these two features are provided in (Li and Khudanpur, 2008b).

4. Using the Decoder

To produce a translation output for a test document, one needs to follow the following general five-step procedure.

1. Train a language model using a toolkit such as the SRI LM tools (Stolcke, 2002);
2. Extract a translation grammar for the test set. This step itself involves several sub-steps, e.g. preparing a bilingual corpus, obtaining word alignments with a tool like GIZA (Och and Ney, 2003), and extracting the grammar using the suffix-array infrastructure;
3. Find optimal weights for combining the different models and feature functions by using MERT or another training procedure;
4. Write the decoder's configuration file, specifying the language model, translation model, feature weights, and other options. The integrated MERT, when given an initial configuration file, will produce a modified configuration with the final weights. Table 1 shows an example configuration file.
5. Finally, run the decoder to produce the k best translations for each sentence in the test document. For an input file, `test.in`, an output k-best file, `test.kbest`, and a configuration file, `config`, the decoder can be invoked with:

   ```
   java joshua.JoshuaDecoder config test.in test.kbest
   ```

   Often it is helpful to pass additional flags to the JVM to specify the minimum and maximum size of the heap, to adjust the minimum free-heap ratio, or to enable 64-bit mode.

5. Experimental Results

In this section, we evaluate the performance of our decoder on a large-scale Chinese to English translation task.[3]

5.1. System Training

We use various parallel text corpora distributed by the Linguistic Data Consortium (LDC) for the NIST MT evaluation. The parallel data we select contains about 570K Chinese-English sentence pairs, adding up to about 19M words on each side. To train the English language

[3] Again, most of the descriptions here are adopted from (Li and Khudanpur, 2008b).

```
# lm file location
lm_file=example.trigram.lm.gz

# tm file location
tm_file=example.hiero.tm.gz

# lm model weight
lm 1.000000

# translation model weights
phrasemodel pt 0 1.066893
phrasemodel pt 1 0.752247
phrasemodel pt 2 0.589793

# wordpenalty weight
wordpenalty -2.844814
```

Table 1. An example configuration file. For conciseness, this file neglects some standard configuration options (e.g. k*-best size).*

models, we use the English side of the parallel text and a subset of the English Gigaword corpus, for a total of about 130M words.

We use the GIZA toolkit (Och and Ney, 2003), a suffix-array architecture (Lopez, 2007), the SRILM toolkit (Stolcke, 2002), and minimum error rate training (Och, 2003) to obtain word-alignments, a translation model, language models, and the optimal weights for combining these models, respectively.

5.2. Improvements in Decoding Speed

We use a Python implementation of a state-of-the-art decoder as our baseline[4] for decoder comparisons. For a direct comparison, we use exactly the same models and pruning parameters. The SCFG contains about 3M rules, the 5-gram LM explicitly lists about 49M n-grams, $n = 1, 2, \ldots, 5$, and the pruning uses $\beta = 10$, $b = 30$ and $\epsilon = 0.1$.

As shown in Table 2, the Java decoder (without explicit parallelization) is 22 times *faster* than the Python decoder, while achieving slightly *better* translation quality as measured by BLEU-4 (Papineni et al., 2002). The parallelization further speeds it up by a factor of 1.7, making the parallel Java decoder is 38 times faster than the Python decoder.

We have also used the decoder to successfully decode about one million sentences for a large-scale discriminative training experiment (Li and Khudanpur, 2008a), showing that the

[4]We are extremely thankful to Philip Resnik at University of Maryland for allowing us the use of their Python decoder as the baseline. Thanks also go to David Chiang who originally implemented the decoder.

Decoder	Speed (sec/sent)	BLEU-4	
		MT '03	MT '05
Python	26.5	34.4%	32.7%
Java	1.2	**34.5%**	**32.9%**
Java (parallel)	**0.7**		

Table 2. Decoder Comparison: Translation speed and quality on the 2003 and 2005 NIST MT benchmark tests.

decoder is stable and scalable.

5.3. Impact of a Distributed Language Model

We use the SRILM toolkit to build eight 7-gram language models, and load and call the LMs using a distributed LM architecture as discussed before. As shown in Table 3, the 7-gram distributed language model (DLM) significantly improves translation performance over the 5-gram LM. However, decoding is significantly slower (12.2 sec/sent when using the non-parallel decoder) due to the added network communication overhead.

LM type	# n-grams	MT '03	MT '05
5-gram LM	49 M	34.5%	32.9%
7-gram DLM	310 M	**35.5%**	**33.9%**

Table 3. Distributed language model: the 7-gram LM cannot be loaded alongside the SCFG on a single machine; via distributed computing, it yields significant improvement in BLEU-4 over a 5-gram.

6. Conclusions

We have described a scalable decoder for parsing-based machine translation. It is written in Java and implements all the essential algorithms described in (Chiang, 2007) and (Li and Khudanpur, 2008b): chart-parsing, n-gram language model integration, beam- and cube-pruning, and unique k-best extraction. Additionally, parallel and distributed computing techniques are exploited to make it scalable. We demonstrate that our decoder is 38 times faster than a baseline decoder written in Python, and that the distributed language model is very useful to improve translation quality in a large-scale task. The decoder has been used for decoding millions of sentences for a large-scale discriminative training task (Li and Khudanpur, 2008b).

Acknowledgments

We would like to thank the other Joshua developers for their contributions to the code: Chris Dyer, Lane Schwartz, Jonathan Weese, and Omar Zaidan. We also thank Adam Lopez, Smaranda Muresan and Philip Resnik for very helpful discussions. This research was supported in part by the Defense Advanced Research Projects Agency's GALE program under Contract No. HR0011-06-2-0001 and the National Science Foundation under grants Numbers 0713448 and 0840112. The views and findings are the authors' alone.

Bibliography

Bertoldi, Nicola, Barry Haddow, and Jean-Baptiste Fouet. 2009. Improved minimum error rate training in Moses. *The Prague Bulletin of Mathematical Linguistics*, this volume.

Callison-Burch, Chris, Colin Bannard, and Josh Schroeder. 2005. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-2005)*, Ann Arbor, Michigan.

Chen, Boxing, Deyi Xiong, Min Zhang, Aiti Aw, and Haizhou Li. 2008. I2R multi-pass machine translation system for iwslt 2008. In *Proceedings of the International Workshop on Spoken Language Technology*, Honolulu, Hawaii.

Chiang, David. 2006. An introduction to synchronous grammars. Tutorial available at http://www.isi.edu/~chiang/papers/synchtut.pdf.

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

Eisner, Jason. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-2003)*, Sapporo, Japan.

Galley, Michel, 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 the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL-CoLing-2006)*, Sydney, Australia.

Huang, Liang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the International Workshop on Parsing Technologies*, Vancouver, BC, Canada.

Huang, Liang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, Prague, Czech Republic.

Huang, Liang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of the 7th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2006)*, Cambridge, Massachusetts.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, , and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the ACL-2007 Demo and Poster Sessions*, Prague, Czech Republic.

Li, Zhifei and Sanjeev Khudanpur. 2008a. Large-scale discriminative n-gram language models for statistical machine translation. In *Proceedings of the 8th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2008)*, Honolulu, Hawaii.

Li, Zhifei and Sanjeev Khudanpur. 2008b. A scalable decoder for parsing-based machine translation with equivalent language model state maintenance. In *In Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation*, Colmbus, Ohio.

Liu, Yang, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment templates for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL-CoLing-2006)*, Sydney, Australia.

Lopez, Adam. 2007. Hierarchical phrase-based translation with suffix arrays. In *Proceedings of the Joint Meeting of the Conferences on Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL)*.

Marton, Yuval and Philip Resnik. 2008. Soft syntactic constraints for hierarchical phrased-based translation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Colmbus, Ohio.

Och, Franz Josef. 2003. Minimum error rate training for statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-2003)*, Sapporo, Japan.

Och, Franz Josef and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, Philadelphia, Pennsylvania.

Quirk, Chris, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-2005)*, Ann Arbor, Michigan.

Stolcke, Andreas. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, Denver, Colorado, September.

Talbot, David and Miles Osborne. 2007. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, Prague, Czech Republic.

Venugopal, Ashish and Andreas Zollmann. 2009. Grammar based statistical MT on Hadoop: An end-to-end toolkit for large scale PSCFG based statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, this volume.

Venugopal, Ashish, Andreas Zollmann, and Stephan Vogel. 2007. An efficient two-pass approach to synchronous-CFG driven statistical MT. In *Proceedings of the Human Language Technology Conference of the North American chapter of the Association for Computational Linguistics (HLT/NAACL-2007)*, Rochester, New York,.

Zaidan, Omar F. 2009. Z-MERT: A fully configurable open source tool for minimum error rate training of machine translation systems. *The Prague Bulletin of Mathematical Linguistics*, this volume.