Joshua 3.0: Syntax-based Machine Translation with the Thrax Grammar Extractor

Jonathan Weese[1], **Juri Ganitkevitch**[1], **Chris Callison-Burch**[1], **Matt Post**[2] and **Adam Lopez**[1,2]
[1]Center for Language and Speech Processing
[2]Human Language Technology Center of Excellence
Johns Hopkins University

Abstract

We present progress on Joshua, an open-source decoder for hierarchical and syntax-based machine translation. The main focus is describing Thrax, a flexible, open source synchronous context-free grammar extractor. Thrax extracts both hierarchical (Chiang, 2007) and syntax-augmented machine translation (Zollmann and Venugopal, 2006) grammars. It is built on Apache Hadoop for efficient distributed performance, and can easily be extended with support for new grammars, feature functions, and output formats.

1 Introduction

Joshua is an open-source[1] toolkit for hierarchical machine translation of human languages. The original version of Joshua (Li et al., 2009) was a reimplementation of the Python-based Hiero machine-translation system (Chiang, 2007); it was later extended (Li et al., 2010) to support richer formalisms, such as SAMT (Zollmann and Venugopal, 2006).

The main focus of this paper is to describe this past year's work in developing *Thrax* (Weese, 2011), an open-source grammar extractor for Hiero and SAMT grammars. Grammar extraction has shown itself to be something of a black art, with decoding performance depending crucially on a variety of features and options that are not always clearly described in papers. This hindered direct comparison both between and within grammatical formalisms. Thrax standardizes Joshua's grammar extraction procedures by providing a flexible and configurable means of specifying these settings. Section 3 presents a systematic comparison of the two grammars using identical feature sets.

In addition, Joshua now includes a single parameterized script that implements the entire MT pipeline, from data preparation to evaluation. This script is built on top of a module called *CachePipe*. CachePipe is a simple wrapper around shell commands that uses SHA-1 hashes and explicitly-provided lists of dependencies to determine whether a command needs to be run, saving time both in running and debugging machine translation pipelines.

2 Thrax: grammar extraction

In modern machine translation systems such as Joshua (Li et al., 2009) and cdec (Dyer et al., 2010), a translation model is represented as a synchronous context-free grammar (SCFG). Formally, an SCFG may be considered as a tuple

$$(N, S, T_\sigma, T_\tau, G)$$

where N is a set of nonterminal symbols of the grammar, $S \in N$ is the goal symbol, T_σ and T_τ are the source- and target-side terminal symbol vocabularies, respectively, and G is a set of *production rules* of the grammar.

Each rule in G is of the form

$$X \to \langle \alpha, \gamma, \sim \rangle$$

where $X \in N$ is a nonterminal symbol, α is a sequence of symbols from $N \cup T_\sigma$, γ is a sequence of

[1]http://github.com/joshua-decoder/joshua

symbols from $N \cup T_\tau$, and $\sim$ is a one-to-one correspondence between the nonterminal symbols of α and γ.

The language of an SCFG is a set of ordered pairs of strings. During decoding, the set of candidate translations of an input sentence f is the set of all e such that the pair (f, e) is licensed by the translation model SCFG. Each candidate e is generated by applying a sequence of production rules $(r_1 \dots r_n)$. The cost of applying each rule is:

$$w(X \rightarrow \langle \alpha, \gamma \rangle) = \prod_i \phi_i(X \rightarrow \langle \alpha, \gamma \rangle)^{\lambda_i} \quad (1)$$

where each ϕ_i is a *feature function* and λ_i is the weight for ϕ_i. The total *translation model score* of a candidate e is the product of the rules used in its derivation. This translation model score is then combined with other features (such as a language model score) to produce an overall score for each candidate translation.

2.1 Hiero and SAMT

Throughout this work, we will reference two particular SCFG types known as Hiero and Syntax-Augmented Machine Translation (SAMT).

A Hiero grammar (Chiang, 2007) is an SCFG with only one type of nonterminal symbol, traditionally labeled X. A Hiero grammar can be extracted from a parallel corpus of word-aligned sentence pairs as follows: If (f_i^j, e_k^l) is a sub-phrase of the sentence pair, we say it is *consistent* with the pair's alignment if none of the words in f_i^j are aligned to words outside of e_k^l, and vice-versa. The consistent sub-phrase may be extracted as an SCFG rule. Furthermore, if a consistent phrase is contained within another one, a hierarchical rule may be extracted by replacing the smaller piece with a nonterminal.

An SAMT grammar (Zollmann and Venugopal, 2006) is similar to a Hiero grammar, except that the nonterminal symbol set is much larger, and its labels are derived from a parse tree over either the source or target side in the following manner. For each rule, if the target side is spanned by one constituent of the parse tree, we assign that constituent's label as the nonterminal symbol for the rule. Otherwise, we assign an extended category of the form $C_1 + C_2$, C_1/C_2, or $C_2 \backslash C_1$ — indicating that the target side spans two adjacent constituents, is a C_1 missing a C_2 to the right, or is a C_1 missing a C_2 on the left, respectively. Table 1 contains a list of Hiero and SAMT rules extracted from the training sentence pair in Figure 1.

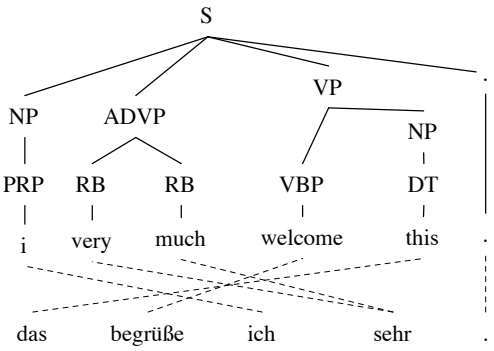

Figure 1: An aligned sentence pair.

2.2 System overview

The following were goals in the design of Thrax:

- the ability to extract different SCFGs (such as Hiero and SAMT), and to adjust various extraction parameters for the grammars;
- the ability to easily change and extend the feature sets for each rule
- scalability to arbitrarily large training corpora.

Thrax treats the grammar extraction and scoring as a series of dependent Hadoop jobs. Hadoop (Venugopal and Zollmann, 2009) is an implementation of Google's MapReduce (Dean and Ghemawat, 2004), a framework for distributed processing of large data sets. Hadoop jobs have two parts. In the *map* step, a set of key/value pairs is mapped to a set of intermediate key/value pairs. In the *reduce* step, all intermediate values associated with an intermediate key are merged.

The first step in the Thrax pipeline is to extract all the grammar rules. The map step in this job takes as input word-aligned sentence pairs and produces a set of ordered pairs (r, c) where r is a rule and c is the number of times it was extracted. During the reduce step, these rule counts are summed, so the result is a set of rules, along with the total number of times each rule was extracted from the entire corpus.

Span	Hiero	SAMT
$[1,3]$	$X \rightarrow \langle$sehr, very much$\rangle$	$ADVP \rightarrow \langle$sehr, very much$\rangle$
$[0,3]$	$X \rightarrow \langle X$ sehr, X very much$\rangle$	$PRP + ADVP \rightarrow \langle PRP$ sehr, PRP very much$\rangle$
$[3,4]$	$X \rightarrow \langle$begrüße, welcome$\rangle$	$VBP \rightarrow \langle$begrüße, welcome$\rangle$
$[0,6]$	$X \rightarrow \langle X$ ich sehr ., i very much X .$\rangle$	$S \rightarrow \langle VP$ ich sehr ., i very much VP .$\rangle$
$[0,6]$	$X \rightarrow \langle X$., X .$\rangle$	$S \rightarrow \langle S/.$., $S/.$.$\rangle$

Table 1: A subset of the Hiero and SAMT rules extracted from the sentence pair of Figure 1.

Given the rules and their counts, a separate Hadoop job is run for each feature. These jobs can all be submitted at once and run in parallel, avoiding the linear sort-and-score workflow. The output from each feature job is the same set of pairs (r, c) as the input, except each rule r has been annotated with some feature score f.

After the feature jobs have been completed, we have several copies of the grammar, each of which has been scored with one feature. A final Hadoop job combines all these scores to produce the final grammar.

Some users may not have access to a Hadoop cluster. Thrax can be run in standalone or pseudo-distributed mode on a single machine. It can also be used with Amazon Elastic MapReduce,[2] a web service that provides computation time on a Hadoop cluster on-demand.

2.3 Extraction

The first step in the Thrax workflow is the extraction of grammar rules from an input corpus. As mentioned above, Hiero and SAMT grammars both require a parallel corpus with word-level alignments. SAMT additionally requires that the target side of the corpus be parsed.

There are several parameters that can make a significant difference in a grammar's overall translation performance. Each of these parameters is easily adjustable in Thrax by changing its value in a configuration file.

- maximum rule span
- maximum span of consistent phrase pairs
- maximum number of nonterminals
- minimum number of aligned terminals in rule
- whether to allow adjacent nonterminals on source side
- whether to allow unaligned words at the edges of consistent phrase pairs

Chiang (2007) gives reasonable heuristic choices for these parameters when extracting a Hiero grammar, and Lopez (2008) confirms some of them (maximum rule span of 10, maximum number of source-side symbols at 5, and maximum number of nonterminals at 2 per rule). Zollmann et al. (2008) provided comparisons among phrase-based, hierarchical, and syntax-based models, but did not report extensive experimentation with the model parameterizations.

When extracting Hiero- or SAMT-style grammars, the first Hadoop job in the Thrax workflow takes in a parallel corpus and produces a set of rules. But in fact Thrax's extraction mechanism is more general than that; all it requires is a function that maps a string to a set of rules. This makes it easy to implement new grammars and extract them using Thrax.

2.4 Feature functions

Thrax considers feature functions of two types: first, there are features that can be calculated by looking at each rule in isolation. Such features do not require a Hadoop job to calculate their scores, since we may inspect the rules in any order. (In practice, we calculate the scores at the very last moment before outputting the final grammar.) We call these features *simple features*. Thrax implements the following simple features:

- a binary indicator functions denoting:
  - whether the rule is purely abstract (i.e., has no terminal symbols)

[2] `http://aws.amazon.com/elasticmapreduce/`

  - the rule is purely lexical (i.e., has no nonterminals)
  - the rule is monotonic or has reordering
  - the rule has adjacent nonterminals on the source side

- counters for
  - the number of unaligned words in the rule
  - the number of terminals on the target side of the rule

- a constant phrase penalty

In addition to simple features, Thrax also implements *map-reduce features*. These are features that require comparing rules in a certain order. Thrax uses Hadoop to sort the rules efficiently and calculate these feature functions. Thrax implements the following map-reduce features:

- Phrasal translation probabilities $p(\alpha|\gamma)$ and $p(\gamma|\alpha)$, calculated with relative frequency:

$$p(\alpha|\gamma) = \frac{C(\alpha, \gamma)}{C(\gamma)} \quad (2)$$

  (and vice versa), where $C(\cdot)$ is the number of times a given event was extracted.

- Lexical weighting $p_{lex}(\alpha|\gamma, A)$ and $p_{lex}(\gamma|\alpha, A)$. We calculate these weights as given in (Koehn et al., 2003): let A be the alignment between α and γ, so $(i, j) \in A$ if and only if the ith word of α is aligned to the jth word of γ. Then we can define $p_{lex}(\gamma|\alpha)$ as

$$\prod_{i=1}^{n} \frac{1}{|\{j : (i, j) \in A\}|} \sum_{(i,j)\in A} w(\gamma_j|\alpha_i) \quad (3)$$

  where α_i is the ith word of α, γ_j is the jth word of γ, and $w(y|x)$ is the relative frequency of seeing word y given x.

- Rarity penalty, given by

$$\exp(1 - C(X \rightarrow \langle \alpha, \gamma \rangle)) \quad (4)$$

  where again $C(\cdot)$ is a count of the number of times the rule was extracted.

The above features are all implemented and can be turned on or off with a keyword in the Thrax configuration file.

It is easy to extend Thrax with new feature functions. For simple features, all that is needed is to implement Thrax's SIMPLEFEATURE interface defining a method that takes in a rule and calculates a feature score. Map-reduce features are slightly more complex: to subclass MAPREDUCEFEATURE, one must define a mapper and reducer, but also a sort comparator to determine in what order the rules are compared during the reduce step.

2.5 Related work

Joshua includes a simple Hiero extractor (Schwartz and Callison-Burch, 2010). The extractor runs as a single Java process, which makes it difficult to extract larger grammars, since the host machine must have enough memory to hold all of the rules at once. Joshua's extractor scores each rule with three feature functions — lexical probabilities in two directions, and one phrasal probability score $p(\gamma|\alpha)$.

The SAMT implementation of Zollmann and Venugopal (2006) includes a several-thousand-line Perl script to extract their rules. In addition to phrasal and lexical probabilities, this extractor implements several other features that are also described in section 2.4.

Finally, the cdec decoder (Dyer et al., 2010) includes a grammar extractor that performs well only when all rules can be held in memory.

Memory usage is a limitation of both the Joshua and cdec extractors. Translation models can be very large, and many feature scores require accumulation of statistical data from the entire set of extracted rules. Since it is impractical to keep the entire grammar in memory, rules are usually sorted on disk and then read sequentially. Different feature calculations may require different sort orders, leading to a linear workflow that alternates between sorting the grammar and calculating a feature score. To calculate more feature scores, more sorts have to be performed. This discourages the implementation of new features. For example, Joshua's built-in rule extractor calculates the phrasal probability $p(\gamma|\alpha)$ for each rule but, to save time, does not calculate its obvious counterpart $p(\alpha|\gamma)$, which would require another sort.

Language pair	sentences (K)	words (M)
cs–en	332	4.7
de–en	279	5.5
en–cs	487	6.9
en–de	359	7.2
en–fr	682	12.5
fr–en	792	14.4

Table 2: Training data size after subsampling.

pair	hiero	SAMT	improvement
cz-en	21.1	21.7	+0.6
en-cz	16.8	16.9	+0.1
de-en	18.9	19.5	+0.6
en-de	14.3	14.9	+0.6
fr-en	28.0	-	-
en-fr	30.4	-	-

Table 3: Single-reference BLEU-4 scores.

The SAMT extractor does not have a problem with large data sets; SAMT can run on Hadoop, as Thrax does.

The Joshua and cdec extractors only extract Hiero grammars, and Zollmann and Venugopal's extractor can only extract SAMT-style grammars. They are not designed to score arbitrary feature sets, either. Since variation in translation models and feature sets can have a significant effect on translation performance, we have developed Thrax in order to make it easy to build and test new models.

3 Experiments

We built systems for six language pairs for the WMT 2011 shared task: cz-en, en-cz, de-en, en-de, fr-en, and en-fr.[3] For each language pair, we built both SAMT and hiero grammars.[4] Table 3 contains the results on the complete WMT 2011 test set.

To train the translation models, we used the provided Europarl and news commentary data. For cz-en and en-cz, we also used sections of the CzEng parallel corpus (Bojar and Žabokrtský, 2009). The parallel data was subsampled using Joshua's built-in subsampler to select sentences with n-grams relevant to the tuning and test set. We used SRILM to train a 5-gram language model with Kneser-Ney smoothing using the appropriate side of the parallel data. For the English LM, we also used English Gigaword Fourth Edition.[5]

Before extracting an SCFG with Thrax, we used the provided Perl scripts to tokenize and normalize the data. We also removed any sentences longer than 50 tokens (after tokenization). For SAMT grammar extraction, we parsed the English training data using the Berkeley Parser (Petrov et al., 2006) with the provided Treebank-trained grammar.

We tuned the model weights against the WMT08 test set (`news-test2008`) using Z-MERT (Zaidan, 2009), an implementation of minimum error-rate training included with Joshua. We decoded the test set to produce a 300-best list of unique translations, then chose the best candidate for each sentence using Minimum Bayes Risk reranking (Kumar and Byrne, 2004). Figure 2 shows an example derivation with an SAMT grammar. To re-case the 1-best test set output, we trained a true-case 5-gram language model using the same LM training data as before, and used an SCFG translation model to translate from the lowercased to true-case output. The translation model used rules limited to five tokens in length, and contained no hierarchical rules.

4 CachePipe: Cached pipeline runs

Machine translation pipelines involve the specification and execution of many different datasets, training procedures, and pre- and post-processing techniques that can have large effects on translation outcome, and which make direct comparisons between systems difficult. The complexity of managing these pipelines and experimental environments has led to a number of different experimental management systems, such as Experiment.perl,[6] Joshua 2.0's Makefile system (Li et al., 2010), and LoonyBin (Clark and Lavie, 2010). In addition to managing the pipeline, these scripts employ different techniques to avoid expensive recomputation by caching steps. However, these approaches are based on simple but

[3] fr=French, cz=Czech, de=German, en=English.

[4] Except for fr-en and en-fr. We were unable to decode with SAMT grammars for these language pairs due to their large size. We have since resolved this issue and will have scores for the final version of the paper.

[5] LDC2009T13

[6] `http://www.statmt.org/moses/?n=FactoredTraining.EMS`

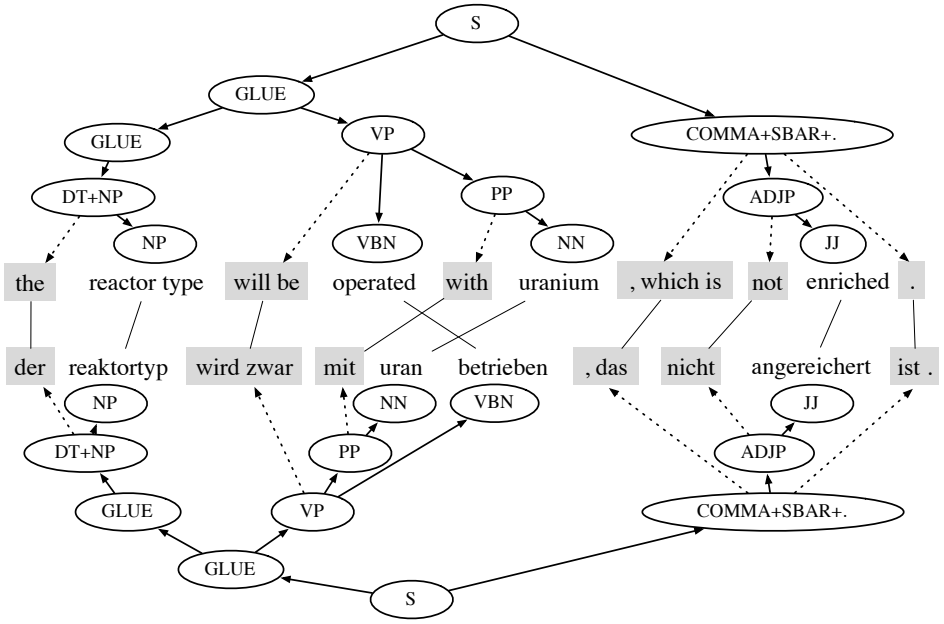

Figure 2: An SAMT derivation. The shaded terminal symbols are the lexicalized part of a rule with terminals and non-terminals. The unshaded terminals are directly dominated by a nonterminal symbol.

unreliable heuristics (such as timestamps or file existence) to make the caching determination.

Our solution to the caching dependency problem is CachePipe. CachePipe is designed with the following goals: (1) robust content-based dependency checking and (2) ease of use, including minimal editing of existing scripts. CachePipe is essentially a wrapper around command invocations. Presented with a command to run and a list of file dependencies, it computes SHA-1 hashes of the dependencies and of the command invocation and stores them; the command is executed only if any of those hashes are different from previous runs. A basic invocation involves specifying (1) a name or identifier associated with the command or step, (2) the command to run, and (3) a list of file dependencies. For example, to copy file `a` to `b` from a shell prompt, the following command could be used:

```
cachecmd copy "cp a b" a b
```

The first time the command is run, the file would be copied; afterwards, the command would be skipped after CachePipe verified that the contents of the dependencies `a` and `b` had not changed.

CachePipe is open-source software, distributed with Joshua or available separately.[7] It currently provides both a shell script interface and a programmatic API for Perl. It accepts a number of other arguments and dependency types. It also serves as the foundation of a new script in Joshua 3.0 that implements the complete Joshua pipeline, from data preparation to evaluation.

5 Future work

Thrax is currently limited to SCFG-based translation models. A natural development would be to extract GHKM grammars (Galley et al., 2004) or more recent tree-to-tree models (Zhang et al., 2008; Liu et al., 2009; Chiang, 2010). We also hope that Thrax will continue to be extended with more feature functions as researchers develop and contribute them.

Acknowledgements

This research was supported by in part by the EuroMatrixPlus project funded by the European Commission (7th Framework Programme), and by the NSF under grant IIS-0713448. Opinions, interpretations, and conclusions are the authors' alone.

[7] `https://github.com/joshua-decoder/cachepipe`

References

Ondřej Bojar and Zdeněk Žabokrtský. 2009. CzEng0.9: Large Parallel Treebank with Rich Annotation. *Prague Bulletin of Mathematical Linguistics*, 92. in print.

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

David Chiang. 2010. Learning to translate with source and target syntax. In *Proc. ACL*, Uppsala, Sweden, July.

Jonathan H. Clark and Alon Lavie. 2010. Loonybin: Keeping language technologists sane through automated management of experimental (hyper) workflows. In *Proc. LREC*.

Jeffrey Dean and Sanjay Ghemawat. 2004. Mapreduce: Simplified data processing on large clusters. In *OSDI*.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proc. ACL 2010 System Demonstrations*, pages 7–12.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proc. NAACL*, Boston, Massachusetts, USA, May.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proc. NAACL*, Morristown, NJ, USA.

Shankar Kumar and William Byrne. 2004. Minimum bayes-risk decoding for statistical machine translation. In *Proc. NAACL*, Boston, Massachusetts, USA, May.

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 *Proc. WMT*, 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 *Proc. WMT*.

Yang Liu, Yajuan Lü, and Qun Liu. 2009. Improving tree-to-tree translation with packed forests. In *Proc. ACL*, Suntec, Singapore, August.

Adam Lopez. 2008. Tera-scale translation models via pattern matching. In *Proc. COLING*, Manchester, UK, August.

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

Lane Schwartz and Chris Callison-Burch. 2010. Hierarchical phrase-based grammar extraction in joshua: Suffix arrays and prefix trees. *The Prague Bulletin of Mathematical Linguistics*, 93:157–166, January.

Ashish Venugopal and Andreas Zollmann. 2009. Grammar based statistical MT on Hadoop: An end-to-end toolkit for large scale PSCFG based MT. *The Prague Bulletin of Mathematical Linguistics*, 91:67–78.

Jonathan Weese. 2011. A systematic comparison of synchronous context-free grammars for machine translation. Master's thesis, Johns Hopkins University, May.

Omar F. Zaidan. 2009. Z-MERT: A fully configurable open source tool for minimum error rate training of machine translation systems. *The Prague Bulletin of Mathematical Linguistics*, 91:79–88.

Min Zhang, Hongfei Jiang, Aiti Aw, Haizhou Li, Chew Lim Tan, and Sheng Li. 2008. A tree sequence alignment-based tree-to-tree translation model. In *Proc. ACL*, Columbus, Ohio, June.

Andreas Zollmann and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *Proc. NAACL Workshop on Statistcal Machine Translation*, New York, New York.

Andreas Zollmann, Ashish Venugopal, Franz Josef Och, and Jay Ponte. 2008. A systematic comparison of phrase-based, hierarchical and syntax-augmented statistical mt. In *Proc. COLING*.