PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 95 APRIL 2011 5–18

A Guide to Jane, an Open Source Hierarchical Translation Toolkit

Daniel Stein, David Vilar, Stephan Peitz, Markus Freitag, Matthias Huck, Hermann Ney

Chair for Computer Science 6, RWTH Aachen University

Abstract

Jane is RWTH's hierarchical phrase-based translation toolkit. It includes tools for phrase extraction, translation and scaling factor optimization, with efficient and documented programs of which large parts can be parallelized. The decoder features syntactic enhancements, reorderings, triplet models, discriminative word lexica, and support for a variety of language model formats. In this article, we will review the main features of Jane and explain the overall architecture. We will also indicate where and how new models can be included.

1. Introduction

This article describes the open source hierarchical phrase-based decoder Jane and its associated toolkit, which was released for non-commerical use in Vilar et al. (2010). Jane follows the hierarchical phrase model as described in Chiang (2007), which can be seen as an extension of the standard phrase model, where the phrases are allowed to have "gaps". In this way, long-distance dependencies and reorderings can be modelled in a consistent way. As in nearly all current statistical approaches to machine translation, this model is embedded in a log-linear model combination.

Jane features syntactic enhancements, additional reorderings, triplet models, discriminative word lexica, and support for a variety of language model formats. The toolkit also implements algorithms for phrase table extraction, translation and scaling factor optimization. Most processes can be parallelized if the Sun Grid Engine is installed. RWTH has been developing this toolkit during the last two years and it was used successfully in numerous machine translation evaluations. It is written in

 Corresponding author: stein@cs.rwth-aachen.de
Cite as: Daniel Stein, David Vilar, Stephan Peitz, Markus Freitag, Matthias Huck, Hermann Ney. A Guide to Jane, an Open Source Hierarchical Translation Toolkit. The Prague Bulletin of Mathematical Linguistics No. 95, 2011, pp. 5–18. doi: 10.2478/v10108-011-0001-6.

C++ with special attention to clean code, extensibility and efficiency, and is available under an open-source non-commercial license.

This article is mainly directed at developers looking for a short overview of the toolkit's architecture. We will briefly review the main features of Jane, with a strong focus on implementation decisions, and we will also describe how and where new extraction and translation models should be implemented by taking advantage of existing classes. For a more general description, we refer to Vilar et al. (2010), and for performance results we refer to system descriptions of international evaluations, e.g. Heger et al. (2010). Note that an in-depth manual is included in the Jane package as well.

The article is structured as follows: we review the tool invocation in Section 2. Then, we review the main features that are implemented and how they can be extended, first for the extraction (Section 3), then for the decoding (Section 4). We proceed to mention some other included tools in Section 5. After a short comparison with Joshua in Section 6, we give a short conclusion in Section 7.

1.1. Related Work

Jane implements features presented in previous work, developed both at RWTH and other groups. As we go over the features of the system we will provide the corresponding references. It is not the first system of its kind, although it provides some unique features. There are other open source hierarchical decoders available, one of them being SAMT (Zollmann and Venugopal, 2006). The current version is oriented towards Hadoop usage, the documentation is however still missing. Joshua (Li et al., 2009, 2010) is a decoder written in Java by the John Hopkins University. This project is the most similar to our own, however both were developed independently and each one has some unique features. Moses (Koehn et al., 2007) is the de-facto standard phrase-based translation decoder and has now been extended to support hierarchical translation.

2. Invocation

The main core of Jane has been implemented in C++. It is mainly directed at linux systems, and uses SCons (`http://www.scons.org`) as its build system. Prerequisites for some tools are the SRI language model (Stolcke, 2002), cppunit and the Numerical Recipes (Press et al., 2002). Upon building, a variety of programs and scripts are created to offer a flexible extraction, translation and optimization framework. Alignment training is not included, since well established tools for this purpose exist. Jane accepts most common alignment formats.

In general, all tools support the option `--help` which outputs a compact description of the available command line options. Some programs also support `--man` for a more verbose description in the form of a Unix man page. These manual pages are

generated automatically and thus are always up-to-date. Nearly all components accept a `--verbosity` parameter for controlling the amount of information they report. The parameter can have 6 possible values, ranging from `noLog` to `insaneDebug`.

The options have a hierarchical structure, and the more complex modules within a larger tool typically have their own naming space. For example, the size of the internal n-best list in the cube prune algorithm is set with `--CubePrune.generationNbest`, and the file for the language model can be set with `--CubePrune.LM.file`. We refer to each of these sections as *components*. There are components for the search algorithms, for the language models, for the phrase table, et cetera. The name can also be replaced by a wildcard (*) if all components are to be addressed.

Although all of the options can be specified in the command line, a config file can be used in order to avoid repetitive typing, by invoking the program with `--config <config-file>`. Options specified in the command line have precedence over the config file.

If the Sun Grid Engine (`http://www.sun.com/software/sge/`) is available, nearly all operations of Jane can be parallelized. For the extraction process, the corpus is split into chunks (the granularity being user-controlled) which are distributed in the computer cluster. Count collection, marginal computation and count normalization all happen in an automatic and parallel manner. For the translation process, a batch job is started on a number of computers. A server distributes the sentences to translate to the computers that have been made available to the translation job. Additionally, for the minimum error rate training methods, random restarts may be performed on different computers in a parallel fashion.

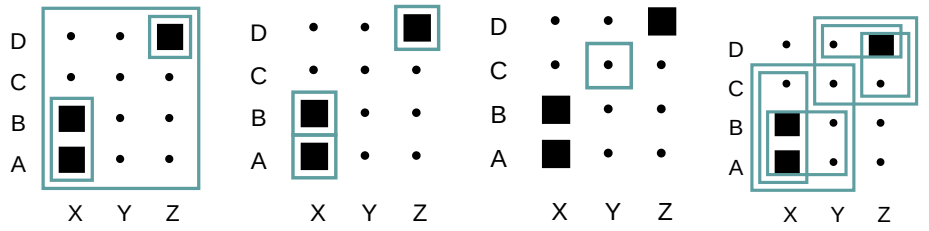
The same client-server infrastructure used for parallel translation may also be reused for interactive systems. Although no code in this direction is provided, one would only need to implement a corresponding frontend which communicates with the translation server (which may be located on another machine).

3. Extraction

In the extraction process, for every training source sentence f_1^J, target sentence e_1^I and alignment $\mathcal{A}$ we generate a set of phrases. First, we extract the set of initial phrases, as defined for the standard phrase-based approach:

$$\begin{aligned} \mathcal{BP}(f_1^J, e_1^I, \mathcal{A}) := \{ \langle f_{j_1}^{j_2}, e_{i_1}^{i_2} \rangle \mid j_1, j_2, i_1, i_2 \qquad & \text{so that} \\ \forall (j,i) \in \mathcal{A} : (j_1 \le j \le j_2 \Leftrightarrow i_1 \le i \le i_2) \qquad & (1) \\ \wedge \exists (j,i) \in \mathcal{A} : (j_1 \le j \le j_2 \wedge i_1 \le i \le i_2) \} . & \end{aligned}$$

See Figure 1(a) for an example of a valid lexical phrase. Words that remain unaligned in the corpus might be problematic for the translation if at translation time they appear in different contexts as in the training corpus. They are not treated as



(a) Valid lexical blocks (b) Single Word Heuristic (c) Forced Single Word Heuristic (d) Extraction of Non-Aligned Word Pairs

Figure 1. Extraction heuristics applied for initial phrases

unknown words, as Jane has them in its internal vocabulary, but is unable to produce a valid parse in the decoding step, which is why we allow for three heuristics in the extraction procedure. In the *single word* heuristic, all single-word pairs derived from each alignment are extracted, even if they would not consist of a valid phrase based on Equation 1 (Figure 1(b)). In the *forced single word* heuristic (Figure 1(c)), all word pairs that are neither covered by a source nor a target alignment are extracted as additional word pairs . Finally, in the *non-aligned heuristic*, all initial phrases are also extended whenever there are non-aligned words on the phrase border (Figure 1(d)).

After the extraction of the lexical phrases, we look for those phrases that contain smaller sub-phrases to extract hierarchical phases. The smaller phrases are suppressed and gaps are produced on the larger ones. For computing probabilities, we compute the counts of each phrase and normalize them, i.e. compute their relative frequencies. Note that the heuristics mentioned above are typically restricted from forming hierarchical phrases, since for some corpora we would produce an arbitrary large number of entries, but this behaviour can be controlled through run-time options. Also due to phrase table size considerations, we typically filter the phrases to only those that are needed for the translation of a given corpus.

Figure 2 shows a schematic representation of the extraction process. In a first pass we extract the bilingual phrases that are necessary for translating the given corpus. For normalization, we still need to compute the marginals for the source and target parts of the phrases. For the source marginals, we can already limit the computation at this stage using the source filter text, but for the target marginals, we do not know in advance which ones will be necessary. Therefore we compute them in a second pass, using the target parts of the bilingual phrases as a target filter.

When parallelizing this operation, the corpus is split into chunks which are distributed in the computer cluster. Count collection, marginal computation and count normalization all happen in an automatic manner.

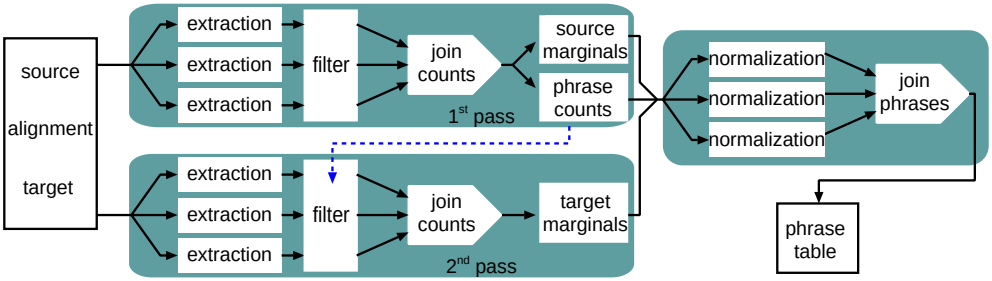


Figure 2. Workflow of the extraction procedure

3.1. Additional Features

Each phrase pair can have an arbitrary number of additional features associated with it. They may hold information about the alignment within the phrase, linguistic information, and more. In order to facilitate the addition of such models, Jane provides two virtual classes, which can be inherited from.

The class `AdditionalExtractionInformation` is a wrapper class for the additional information. The main function that has to be implemented deals with the combination of the feature with another instance of itself. This is necessary whenever the same phrase can be extracted more than once, from different sentence pairs or even in the same sentence pair. Additionally, functions for writing the features into the phrase table have to be provided. In the normalization step, this class is also invoked with the corresponding information about the marginals, in case that it needs this information for producing the final score.

The class `AdditionalExtractionInformationCreator` is responsible for creating instances of AdditionalExtractionInformation. As such, it receives the information available to the extraction process and computes the additional features for the phrase pair. For hierarchical phrases, it may take into account the additional information of the larger phrase and of the phrase that produces the gap in it. In order to assign descriptive labels to the different `AdditionExtractionInformation` classes, new classes must be registered in the corresponding factory functions. The extraction scripts will then automatically perform the necessary steps for generating the additional features, including joining and normalization of the counts.

3.2. Implemented Additional Features

With Jane, it is possible to include numerous additional features in the phrase table. We proceed to review a few of them. The *alignment* information remembers the internal alignment of the extracted phrase. The *dependency* information augments

the phrases with additional dependency-level information in the spirit of Shen et al. (2008). Given a dependency tree at sentence level, we mark phrases that are syntactically valid, and to preserve inner word dependencies. The *heuristic* information marks for each phrase, which of the extraction heuristics as defined in Section 3 have been used to create this particular phrase. The *parse match* information marks whether the phrase matches the yield of a parse tree node, a rather simple approach which has been succesfully applied in Vilar et al. (2008). Finally, the *soft syntactic label* information adds syntactic labels to each phrase and each non-terminal, which are typically derived from a given parse tree. They can be used to compute an additional parse probability based on linguistic experiences, e.g. by emphasizing the need for a verb in the sentence, or by penalizing whenever a noun phrase non-terminal is substituted with a verb phrase, as in Venugopal et al. (2009).

4. Decoding

The search for the best translation proceeds in two steps. First, a monolingual parsing of the input sentence is carried out using the CYK+ algorithm (Chappelier and Rajman, 1998), a generalization of the CYK algorithm which relaxes the requirement for the grammar to be in Chomsky normal form. From the CYK+ chart we extract a hypergraph representing the parsing space.

In a second step the translations are generated from the hypergraph, computing the language model scores in an integrated fashion. Both the cube pruning and cube growing algorithms (Huang and Chiang, 2007) are implemented. For the latter case, the extensions concerning the language model heuristics presented in Vilar and Ney (2009) have also been included.

The majority of the code for both the cube pruning and cube growing algorithms is included in corresponding classes derived from an abstract hypernode class. In this way, the algorithms have access to the hypergraph structure in a natural way. The CYK+ implementation is parametrized in such a way that the derived classes are created as needed. This architecture is highly flexible, and preliminary support for forced alignments in the spirit of Wuebker et al. (2010) is also implemented in this way.

Jane supports four formats for n-gram language models: the ARPA format, the SRI toolkit binary format (Stolcke, 2002), randomized LMs as described in Talbot and Osborne (2007), using the open source implementation made available by the authors of the paper and an in-house, exact representation format. In order to ease the integration of these possibilities, an abstract interface class has been implemented, which provides access to the necessary operations of the language model.

The actual translation procedure is clearly separated from the input/output operations. These are handled in the `RunMode` classes, which are responsible of obtaining the text to translate, calling the translation methods with the appropriate parameters, and writing the result to disk. There are three main RunModes: `SingleBestRunMode` for

single-best operation, `NBestRunMode` for generation of n-best lists and `Optimization-ServerRunMode`. This last one starts a Jane server which offers both single-best and n-best translation functionality, communicating over TCP/IP sockets. In the current implementation this is used for parallel translation and/or optimization in a computer cluster, but it may be easily reused for other applications, like online translation services or interactive machine translation.

For parallelized operation, a series of jobs are started in parallel in the computer cluster. The first one of these jobs is the master and controls the translation process. All Jane processes are started in server mode and wait for translation requests from the master job.

The translation servers are allocated in a dynamic way; if more computers are made available for the translation job, they can be added on the fly to the translation task. The longest sentences are the first ones sent to translate. This simple heuristic performs load balancing, trying to avoid "temporal deadlocks" when the whole array job is just waiting for a computer to finish the translation of a long sentence that happened to be at the end of the corpus. A simple fault-tolerance system is also built-in, which tries to detect if a computer has had problems and resends the associated sentence to another free node. It is however quite basic and although it detects most problems, there are still some cases where non-responding computers may go undetected.

4.1. Additional Models

Jane is designed to be easily extended with new models, added in the log-linear combination. If the new features can be computed at phrase level, the best way is to include them at extraction time, as described in Section 3.1. The decoder can then be instructed to use these additional features at translation time.

For models that cannot computed this way, an abstract `SecondaryModel` class can be derived from. The main function in this class is the `scoreBackpointer` method, which receives a derivation to score, together with a reference to the current hypernode. With this information, the method can obtain all the necessary information for computing the model score. Secondary models implemented this way must be registered in the `SecondaryModelCreator` class. They will be then known to Jane, and facilities for scaling factor allocation, parameter handling and multiple model instantiation will be provided.

There is a limitation to the kind of models that can be implemented this way, namely the models can only influence the search process by generating new scores. No additional information that may change the hypothesis space by e.g. hypothesis recombination is (yet) supported.

4.2. Implemented Additional Models

Like with the additional extraction models, several additional models during the translation are already implemented in Jane using the SecondaryModel infrastructure. In this section we list some of them.

Extended Lexicon Models The Extended Lexicon Models include discriminative lexicon models and triplet models as in Mauser et al. (2009), and are able to take long range dependencies across the whole source sentence into account. The *triplet* model extends the well-known IBM model 1 (Brown et al., 1993), by estimating the probability $p(e|f, f')$ of a target word e based on two source words f, f'. Like IBM 1, the triplet model is trained iteratively using the EM algorithm. During extraction and decoding, f is the source word aligned to the target word e to be translated, while f' ranges over the words in the source sentence. Thus, the second source word f' enables the model to make more informed decisions about translating f into e.

The *discriminative word lexicon* uses the whole source sentence to predict target words, thus taking into account global dependencies. It is modeled as a combination of simple classifiers for each word e from the target vocabulary V_E. Each of these classifiers models whether a certain word e is present in the target sentence ($\delta_e = 1$) or not ($\delta_e = 0$), given the set of source words $\mathbf{f}$. The probability of the target sentence is then modeled as the product of all positive classification probabilities, over all words in the target sentence, times the product of all negative classification probabilities over all words not contained in the target sentence.

Soft Syntactic Labels The Soft Syntactic Labels (cf. Section 3.2) extend the hierarchical model in a similar way as in the work of Venugopal et al. (2009): for every rule in the grammar, we store information about the possible non-terminals that can be substituted in place of the generic non-terminal X, together with a probability for each combination of non-terminal symbols (cf. Figure 3).

During decoding, we compute two additional quantities for each derivation d. The first one is denoted by $p_h(Y|d)$ (h for "head") and reflects the probability that the derivation d under consideration of the additional non-terminal symbols has Y as its starting symbol. This quantity is needed for computing the probability $p_{syn}(d)$ that the derivation conforms with the extended set of non-terminals.

Dependency With the Dependency model, we are able to introduce language models that span over longer distances than shallow n-gram language models. In Figure 4, we can for example evaluate the left-handed dependency of the structure "In", followed by "industry", on the structure "faced". For this, we employ a simple language model trained on dependency structures and compute the probability for the trigram "In industry faced-as-head".

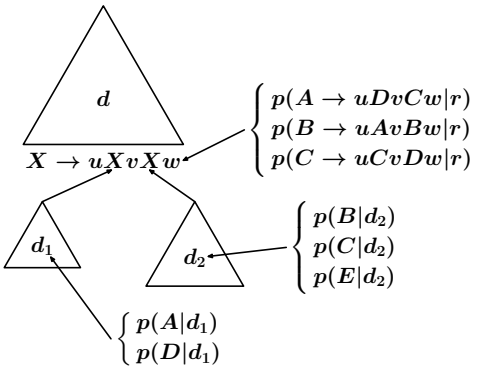


Figure 3. Visualization of the soft syntactic labels approach. For each derivation, the probabilities of non-terminal labels are computed.

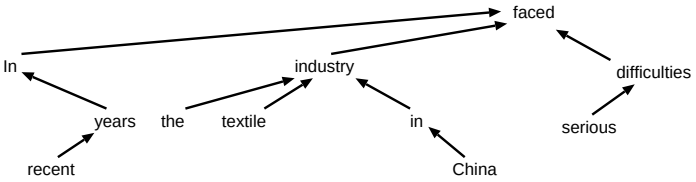


Figure 4. Dependency parsing for the sentence "In recent years, the textile industry in China faced serious difficulties".

The model requires a given dependency tree while extracting the phrase table, and works with this information to derive a tree of the output translation. Note that Shen et al. (2008) only allow for two structures to be extracted: the first possibility is what the authors called a *fixed* dependency structure. With the exception of one word within this phrase, called the *head*, no outside word may have a dependency within this phrase. Also, all inner words may only depend on each other or on the head. For a second structure, called a *floating* dependency structure, the head dependency word may also exist outside the phrase. We do not restrict our algorithms to fixed and floating structures, but rather mark invalid phrases during reconstruction and proceed to reconstruct as much of the dependencies as possible.

Table 1. Speed comparison Jane vs. Joshua.

System	words/sec
Joshua	11.6
Jane cube prune	15.9
Jane cube grow	60.3

5. Other Tools

For optimization of the log-linear scaling factors, we support the minimum error rate training (MERT) described in Och (2003), the MIRA algorithm, applied for machine translation in Chiang et al. (2009), and the downhill simplex method (Nelder and Mead, 1965).

Jane provides a tool to compute the *grow-diag* alignment as presented in Koehn et al. (2003), as well as its alternative as presented in Och and Ney (2003).

6. Comparison with Joshua

As stated in Section 1.1, Joshua (Li et al., 2009) is the most similar decoder to our own, developed in parallel at the Johns Hopkins University.

Jane was started separately and independently. In their basic working mode, both systems implement parsing using a synchronous grammar and include language model information. Each of the projects then progressed independently, and each has unique extension. Efficiency is one of the points where we think Jane outperforms Joshua. One of the reasons can well be the fact that it is written in C++ while Joshua is written in Java. We performed a control experiment on the IWSLT'08 Arabic to English translation taks, using the same settings for both decoders and making sure that the output of both decoders was identical[1]. The speed results can be seen in Table 1. Jane operating with cube prune is nearly 50% faster than Joshua and the speed difference can be improved by using cube growing, although with a slight loss in translation performance. This may be interesting for certain applications like e.g. interactive machine translation or online translation services, where the response time is critical and sometimes even more important than a small (and often hardly noticeable) loss in translation quality.

For comparison of translation results, we refer to the results of the last WMT evaluation shown in Table 2. Johns Hopkins University participated in this evaluation using Joshua, the system was trained by its original authors (Schwartz, 2010) and thus can be considered to be fully optimized. RWTH also participated using Jane among

[1] With some minor exceptions, e.g. unknown words.

Table 2. Results for Jane and Joshua in the WMT 2010 evaluation campaign.

	Jane		Joshua	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]
German-English	21.8	69.5	19.5	66.0
English-German	15.7	74.8	14.6	73.8
French-English	26.6	61.7	26.4	61.4
English-French	25.9	63.2	22.8	68.1

other systems. A detailed description of RWTH's submission can be found in Heger et al. (2010). The scores are computed using the official Euromatrix web interface for machine translation evaluation[2].

As can be seen the performance of Jane and Joshua is similar, but Jane generally achieves better results in BLEU, while Joshua has an advantage in terms of TER. Having different systems is always enriching, and particularly as system combination shows great improvements in translation quality, having several alternative systems can only be considered a positive situation.

7. Conclusion

In this work, we described how and where new models can be integrated into the Jane architecture. We also reviewed the features that are currently implemented. Jane can be downloaded from `http://www.hltpr.rwth-aachen.de/jane`. The toolkit is open-source and free for non-commercial purposes. Other licenses can be negotiated on demand. It is our hope that by adhering to strict code and documentation policies, we enable fellow researchers to adopt and extend the toolkit easily to their needs.

Bibliography

Brown, Peter F., Stephan A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311, June 1993.

Chappelier, Jean-Cédric and Martin Rajman. A Generalized CYK Algorithm for Parsing Stochastic CFG. In *Proc. of the First Workshop on Tabulation in Parsing and Deduction*, pages 133–137, Apr. 1998.

Chiang, David. Hierarchical Phrase-based Translation. *Computational Linguistics*, 33(2): 201–228, June 2007.

[2] `http://matrix.statmt.org/`

Chiang, David, Kevin Knight, and Wei Wang. 11,001 new Features for Statistical Machine Translation. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 218–226, Boulder, Colorado, June 2009.

Heger, Carmen, Joern Wuebker, Matthias Huck, Gregor Leusch, Saab Mansour, Daniel Stein, and Hermann Ney. The RWTH Aachen Machine Translation System for WMT 2010. In *ACL 2010 Joint Fifth Workshop on Statistical Machine Translation and Metrics MATR*, pages 93–97, Uppsala, Sweden, July 2010.

Huang, Liang and David Chiang. Forest Rescoring: Faster Decoding with Integrated Language Models. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 144–151, Prague, Czech Republic, June 2007.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical Phrase-Based Translation. In *Proceedings of the Human Language Technology, North American Chapter of the Association for Computational Linguistics*, pages 54–60, Edmonton, Canada, May 2003.

Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 177–180, Prague, Czech Republic, June 2007.

Li, Zhifei, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Joshua: An Open Source Toolkit for Parsing-Based Machine Translation. In *Proc. of the Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, March 2009. Association for Computational Linguistics.

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 *ACL 2010 Joint Fifth Workshop on Statistical Machine Translation and Metrics MATR*, pages 133–137, Uppsala, Sweden, July 2010.

Mauser, Arne, Saša Hasan, and Hermann Ney. Extending Statistical Machine Translation with Discriminative and Trigger-Based Lexicon Models. In *Proc. of the Conf. on Empirical Methods for Natural Language Processing (EMNLP)*, pages 210–218, Singapore, Aug. 2009.

Nelder, John A. and Roger Mead. The Downhill Simplex Method. *Computer Journal*, 7:308, 1965.

Och, Franz Josef. Minimum Error Rate Training for Statistical Machine Translation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 160–167, Sapporo, Japan, July 2003.

Och, Franz Josef and Hermann Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51, Mar. 2003.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++*. Cambridge University Press, Cambridge, UK, 2002.

Schwartz, Lane. Reproducible Results in Parsing-Based Machine Translation: The JHU Shared Task Submission. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 177–182, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Shen, Libin, Jinxi Xu, and Ralph Weischedel. A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 577–585, Columbus, Ohio, June 2008.

Stolcke, Andreas. SRILM – an Extensible Language Modeling Toolkit. In *Proc. of the Int. Conf. on Spoken Language Processing (ICSLP)*, volume 3, Denver, Colorado, Sept. 2002.

Talbot, David and Miles Osborne. Smoothed Bloom Filter Language Models: Tera-scale LMs on the Cheap. In *Proc. of the Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 468–476, Prague, Czech Republic, June 2007.

Venugopal, Ashish, Andreas Zollmann, N.A. Smith, and Stephan Vogel. Preference Grammars: Softening Syntactic Constraints to Improve Statistical Machine Translation. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 236–244, Boulder, Colorado, June 2009.

Vilar, David and Hermann Ney. On LM Heuristics for the Cube Growing Algorithm. In *Proc. of the Annual Conf. of the European Association for Machine Translation (EAMT)*, pages 242–249, Barcelona, Spain, May 2009.

Vilar, David, Daniel Stein, and Hermann Ney. Analysing Soft Syntax Features and Heuristics for Hierarchical Phrase Based Machine Translation. In *Proc. of the Int. Workshop on Spoken Language Translation (IWSLT)*, pages 190–197, Waikiki, Hawaii, Oct. 2008.

Vilar, David, Daniel Stein, Matthias Huck, and Hermann Ney. Jane: Open source hierarchical translation, extended with reordering and lexicon models. In *Proc. of the Workshop on Statistical Machine Translation*, pages 262–270, Uppsala, Sweden, July 2010.

Wuebker, Joern, Arne Mauser, and Hermann Ney. Training phrase translation models with leaving-one-out. In *48th Annual Meeting of the Association for Computational Linguistics*, pages 475–484, Uppsala, Sweden, 2010.

Zollmann, Andreas and Ashish Venugopal. Syntax Augmented Machine Translation via Chart Parsing. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, New York, June 2006.

Address for correspondence:
Daniel Stein
stein@cs.rwth-aachen.de
Chair for Computer Science 6
RWTH Aachen University
Ahornstr. 55
52056 Aachen, Germany