PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 127–136

Visualizing Data Structures in Parsing-Based Machine Translation

Jonathan Weese, Chris Callison-Burch

Center for Language and Speech Processing, Johns Hopkins University

Abstract

As machine translation (MT) systems grow more complex and incorporate more linguistic knowledge, it becomes more difficult to evaluate independent pieces of the MT pipeline. Being able to inspect many of the intermediate data structures used during MT decoding allows a more fine-grained evaluation of MT performance, helping to determine which parts of the current process are effective and which are not. In this article, we present an overview of the visualization tools that are currently distributed with the Joshua (Li et al., 2009) MT decoder. We explain their use and present an example of how visually inspecting the decoder's data structures has led to useful improvements in the MT model.

1. Introduction

The Joshua machine translation decoder uses a formalism known as synchronous context-free grammars (SCFGs) (Chiang, 2006). A probabilistic SCFG consists of a set of source-language terminal symbols, a set of target-language terminal symbols, a set of nonterminals that is shared between both languages, and set of production rules of the form

$$X \rightarrow \langle \gamma, \alpha, \sim, w \rangle$$

where X is a nonterminal symbol, γ is a (possibly mixed) sequence of nonterminals and source terminals, α is a (again possibly mixed) sequence of nonterminals and target-language terminals, $\sim$ is a one-to-one correspondence between the nonterminals of γ and α, and w is a weight for the production rule.

Using an SCFG to parse the input sentence automatically creates a corresponding target-language sentence. We take the generated sentence as a candidate translation

Corresponding author: jonny@cs.jhu.edu
Cite as: Jonathan Weese, Chris Callison-Burch. Visualizing Data Structures in Parsing-Based Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 127–136. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0016-4.

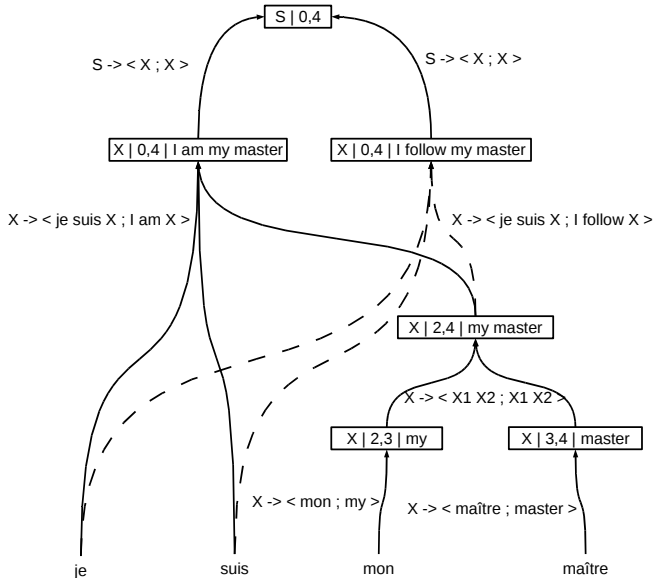

Figure 1. A hypergraph showing two candidate translations of Je suis mon maître.

for the input sentence. The sequence of rules used to generate t_1^m is called a *derivation*. The derivation encodes synchronous parse trees on the source and target sides.

Since we are parsing the input sentence with a probabilistic grammar, we can generate many possible candidate parses for a particular sentence. These parses may share a lot of structure — for example, two different parse trees may contain subtrees that are identical. As part of the search procdeure, the Joshua system compactly stores these structure-sharing trees as a *hypergraph* or *packed forest* where identical subtrees from many parses are represented by a single copy. Structure-sharing hypergraphs are commonly used to represent the result of parsing a sentence with a probabilistic CFG (Billott and Lang, 1989; Klein and Manning, 2001). So it is natural that Joshua's SCFG-based model should use this data structure.

Figure 1 shows an example of a hypergraph with two candidate translations for the French sentence "*Je suis mon maître*." Note that both translations — "I am my master" and "I follow my master" — use the same derivation of "my master" as a phrasal translation for the French *mon maître*. Therefore, the hypergraph maintains only one copy of this shared structure, saving space.

We focus on two data structures used in the decoding process: first, each candidate translation has a *derivation tree* describing how the decoder generated the candidate. Second, there is a *hypergraph* that represents *all* the choices made by the decoder when

translating a single input sentence and shows us the relationships among all the various candidates.

2. Joshua's Visualization Tools

The visualization tools are included in the Joshua Decoder release.[1] They have only one outside dependency: the Java Universal Network/Graph Framework[2] (JUNG), a toolkit for drawing graphs in Java, which is available under the BSD License.

The derivation-tree viewer does not depend on the Joshua decoder. It can visualize any derivation tree as long as the tree representation is identical to Joshua's output. The hypergraph visualizer, on the other hand, depends on the decoding process itself, and therefore on the Joshua decoder.

2.1. Synchronous Derivation Trees

The Joshua decoder includes an option to output text-based representations of the target-side parse tree rather than the plain candidate translation. These textual representations can also be annotated with the source-side span of each nonterminal in the tree. (These outputs can be chosen by setting `use_tree_nbest` and `include_align_index` to true in the Joshua configuration file.) The core of the derivation-tree visualizer takes a string that represents an annotated parse tree and uses JUNG to draw a graph representing the tree.

Using the source-side annotations, the visualizer also draws the parse tree for the input sentence. Since the grammar is synchronous, there is a one-to-one correspondence between nonterminals in the source- and target-side trees. Any differences in the tree structure arise from possible reordering of the nonterminals. The bottom of Figure 2 shows part of the derivation tree generated by Joshua's output of `(ROOT{0-24} ([GOAL]{0-23} ([GOAL]{0-6} ([NP]{0-6} ([NP]{4-6} this meeting) in ([NNP-GPE]{0-1} jerusalem))) ([S]{6-23} ([VP/NP]{14-22} is ([VBG]{19-21} being) considered as ([NP+IN]{14-18} a ([NN]{17-18} show) of ([NN]{15-16} support) for)) ([NP]{6-14} palestinian ([JJ-GPE-ite\NP]{7-14} president ([NP-PERSON]{8-10} mahmoud abbas) ([PP]{10-14} for ([NNP-GPE]{12-13} israel)))) .)))`

The tree visualizer takes the following arguments: a file containing source-side sentences (one per line), a file containing the parallel target-side reference translations, and then one or more files containing n-best translations of the source sentences. Multiple n-best files can be specified in order to contrast different runs of the decoder on the same test set. For example, Figure 3 shows one sentence from the 2009 NIST Urdu–English evaluation, decoded once under a Hiero-style grammar

[1] `http://cs.jhu.edu/~ccb/joshua/`

[2] `http://jung.sourceforge.net/`

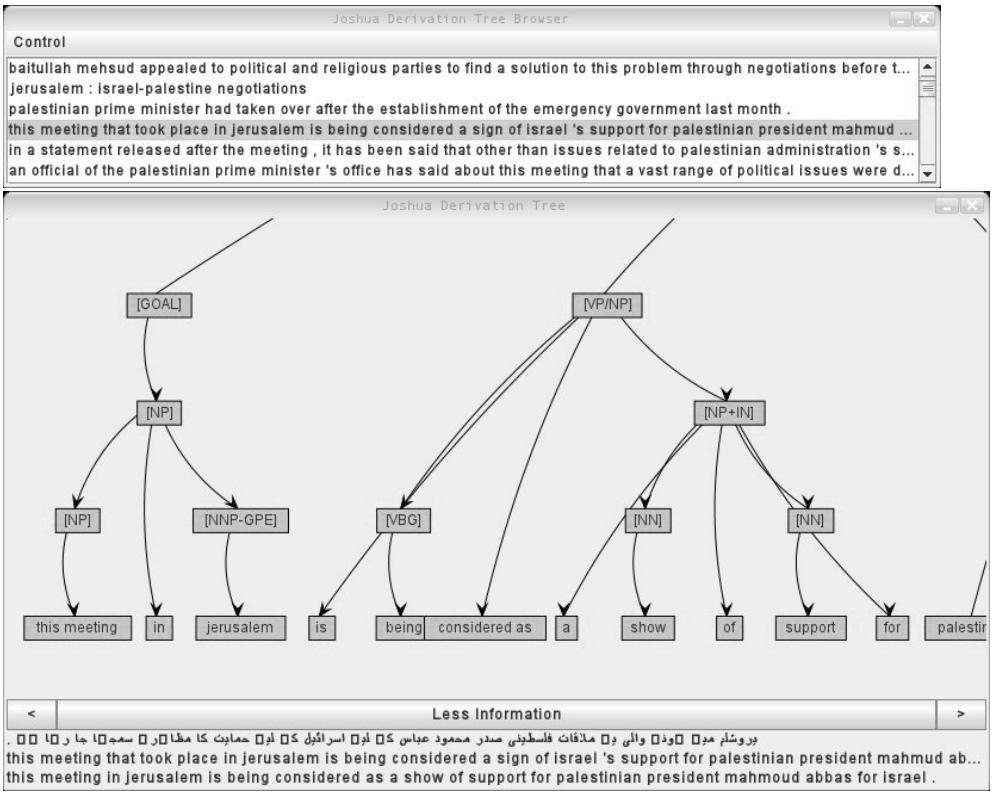

Figure 2. The Derivation Tree browser's sentence selection and tree-viewing windows.

with only one nonterminal symbol X, as introduced in (Chiang, 2005), and once under a syntactically-motivated grammar with a richer nonterminal set as presented in (Zollmann and Venugopal, 2006).

When the visualizer starts up, it creates two types of windows. First there is a window listing all the reference translations. This window is shown on the top of Figure 2. (This is the reason for the reference translation file — an earlier version of the program had the sentences listed by the source side, but this was less useful for users who had no knowledge of the source language.) The second type of window displays a derivation tree.

The browser creates one tree-displaying window (as seen on the bottom of Figure 2) for every n-best file that is passed as an argument. This allows the user to easily compare derivation trees created by different grammars (these are saved in different n-best files, since they're the result of different runs of the decoder). In a tree window,

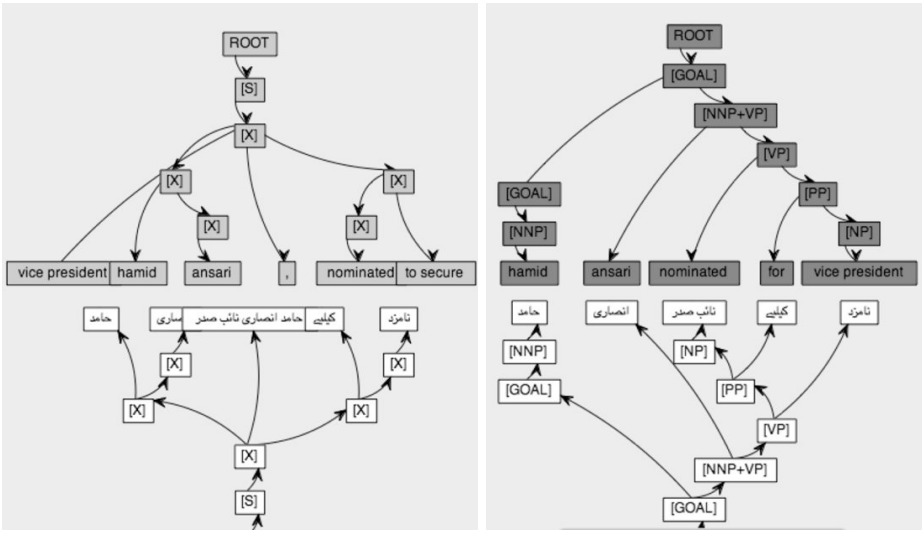

Figure 3. An example visualization of two derivation trees for SCFGs that use a Hiero-style grammar and a syntactically-motivated grammar.

the user can click and drag to inspect different parts of the tree and can use the scroll wheel to zoom in and out.

There are two ways to change trees: clicking the sentence in the list window or using the left- and right-arrow buttons in the tree windows. In either case, all the tree windows are synchronized; that is, whenever the user changes to a different sentence, all tree windows are updated to show the derivation for that sentence. The tree views are anchored; for example, if a user were zoomed in on the first target-side nonterminals of one tree, when he changed to a new sentence and the new tree was displayed, its view would start zoomed in on that same area of the new sentence's tree.

The tree view windows also have a button to give the user more information about the current tree. This button shows the source sentence, the reference translation, and the text of the translation candidate.

2.2. Hypergraphs

The derivation tree viewer is used after running the decoder to inspect its ouput. Since the hypergraph is used during the decoding process and is discarded afterwards, the hypergraph visualizer depends on the decoder itself. Internally, it works by setting a flag to build the graph visualization on-the-fly as the decoder processes the input.

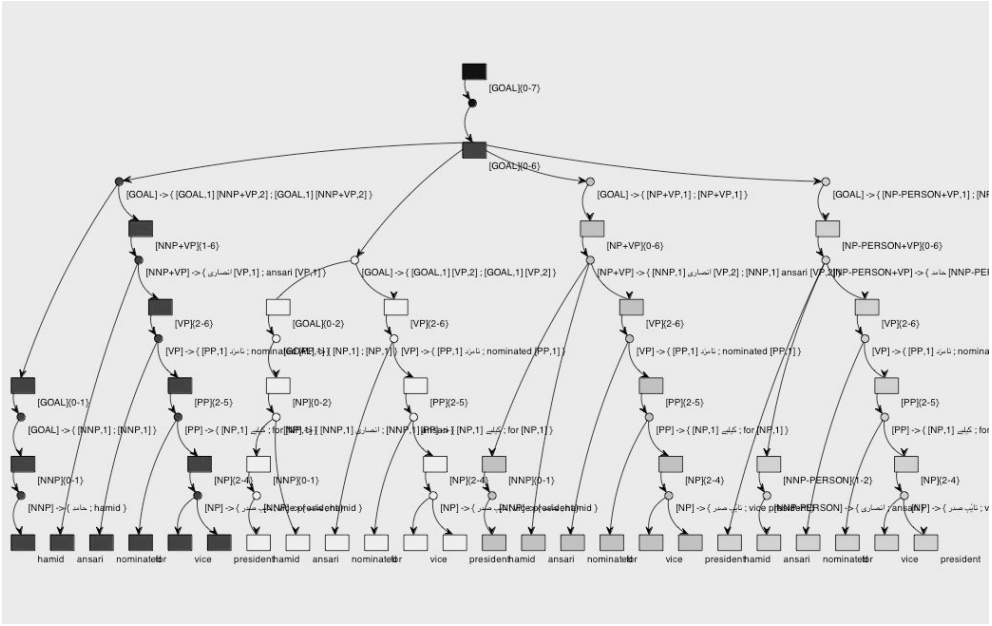

Figure 4. The visualization window for the hypergraph browser.

The command line arguments for the hypergraph visualizer are slightly different from the derivation tree viewer. A reference translation is used in the same way as in the derivation tree viewer — it allows the user to choose a sentence to translate based on the reference translation. Now because the decoder translates source sentences on demand, the other two arguments are a source-sentence file and a Joshua configuration file that should be used for the decoding.

On startup, the hypergraph viewer presents a list of sentences to translate. The reference translations are displayed instead of the source to make it easier for the user. The user selects a sentence from the list and presses the "Decode" button at the bottom. Internally, the hypergraph viewer chooses the associated source sentence and calls Joshua to decode the sentence using the supplied configuration file.

As the hypergraph structure is constructed, the hypergraph viewer uses JUNG to build a corresponding graph. At first glance, the graph that is displayed looks very similar to a derivation tree. In fact, it is a hypergraph representation of the one-best candidate for the source sentence. Recall that the nodes of the hypergraph represent nonterminals or terminals in the derivation of a given source sentence, and the hyperedges represent production rules that are applied at each step. Each node of the hypergraph is a rectangular box in this visualization, and each hyperedge is a small

circle. There may be many possible hyperedges leading from a particular node, each one representing a different rule that could be applied to that node. When only one hyperedge is visualized at each node, the resulting graph represents one candidate translation.

Using the hypergraph viewer, we can inspect the choices that the decoder made at each point. When the user clicks on a node of the hypergraph, a list of all hyperedges leading from that node is displayed on the left. If the user selects a hyperedge from the list, the corresponding subtree is displayed in the graph. A user can even select multiple rules, and all of the resulting subtrees are displayed side-by-side. Figure 4 shows this hypergraph view. We can see four subtrees giving possible derivations of the candidate translation "hamid ansari nominated for vice president."

3. Other Visualization Tools

This section describes two other visualization tools developed by other researchers, and contrasts the goals of the various visualization systems and how the goals are reflected in design choices.

3.1. The Chinese Room

The Chinese Room (Albrecht et al., 2009) is a collaborative translation interface. It uses visualization techniques to allow a user who has no knowledge of the source-side language to collaborate with an MT system to create good translations. This is different from earlier approaches to collaborative MT where the user was often assumed to be a professional translator.

The Chinese Room is designed to allow *translators* (even those with limited or no knowledge of the source language) to produce good translations of the input sentences. Joshua's visualization tools, on the other hand, are designed to help *researchers* debug grammars and improve their translation models. The Chinese Room tries to give the user as much information as possible to create a correct translation. This includes word alignments, glosses for source words from a dictionary, source-side parse structure and so on. All of this information would be useful for a translator. The current tools for Joshua are focused on improving the grammars used in a translation model. We need comparatively less information to gain useful insight into this smaller domain. That is why we have focused only on displaying the derivation trees and hypergraphs.

3.2. DerivTool

DerivTool is a tool for interactively directing the decoding of a sentence using a syntax-based MT model (DeNeefe et al., 2005). Users can choose which rules to apply to a derivation at which time. In the end, the user ends up building up a derivation

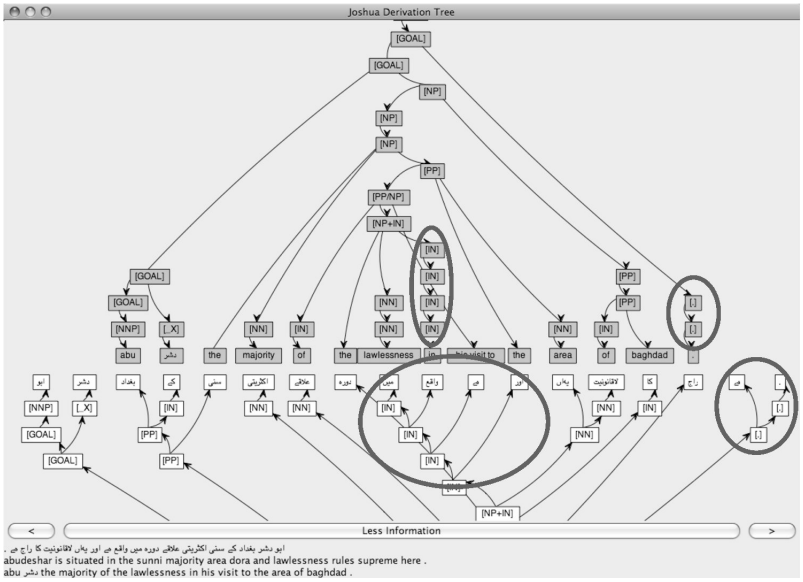

Figure 5. An example of bad production rules that parse pieces of the source sentence without producing any target-side output.

tree for a particular candidate translation. DerivTool is useful for analyzing grammars: a user can immediately tell when a needed rule is missing (they can't continue their intended derivation) and can see which rules are favored by the decoder at a particular point (since rules are displayed in order of frequency).

Joshua's tools and DerivTool both produce derivation trees. The difference is that DerivTool is interactive and Joshua is not. The main advantage of working in batch mode is that it is faster than building up a derivation tree manually step-by-step. We run the decoder seperately, letting it make the translation decisions. Afterwards, the user can evaluate the quality of the trees produced. The Joshua visualization tool can produce many derivation trees all at once (it reads in a file containing the n-best derivations for each sentence of a test set). This makes it easy to compare the different decisions that were possible at decoding time without the user having to manually make the decisions himself.

4. How Joshua's Visualization Tools Help

By visualizing derivation trees for different candidate translations, we can find problems with the SCFGs that underly translations. For example, we used the visual-

ization tool to inspect derivation trees that were produced on the 2009 NIST Urdu–English test set, and noticed that many rules consumed part of the input sentence without producing any output. These rules were used often in the top candidates, but brought down the translation quality. Having discovered that, we manually removed where the target side contained no terminal symbols. This helped improve the translation quality. In Figure 5, we can see rules of the form [IN] $\rightarrow \langle$[IN]γ, [IN]$\rangle$ and [.] $\rightarrow \langle$[.]γ, [.]$\rangle$ being applied.

Pruning grammars of systematically bad production rules is a good way to improve translation quality, and manual inspection of the output to see which grammar rules have been used, and which ones correlate with low translation quality, is an effective way to prune. The derivation tree visualizer helps researchers too notice patterns in rule application among different candidate translations.

Visualizing hypergraphs lets the researcher inspect the decisions that the decoder made when choosing among different rules that could be applied at some point in a derivation. This could be useful, for example, in determining if a particular type of rule is being systematically overweighted or underweighted,

Visualizing the data structures involved in MT decoding allows the researcher to determine empirical rules for improving the grammars involved.

5. Future Work

There are still many improvements that should be made to these visualization tools. We would like to be able to show the terminal alignments that are induced by a particular derivation. But that requires more information than is currently given by the Joshua decoder. It only annotates nonterminals with source-side spans. As an example, consider the French phrase *l'objectif de* X *est de* which corresponds to the English phrase *the goal of* X *is to*. We know that those two phrases have corresponding spans, and that the two X symbols also correspond. But despite this, we don't have word-level alignment information: we don't know if *l'objectif de* corresponds to *the goal of* or to *is to*. Such fine-grained information would be useful for visualizing reordering in MT models.

Another improvement that we think would greatly increase the utility of these visualization tools for research is to add support for exporting the displayed trees as files. It would be nice to be able to save an interesting tree in PDF format so that it could be easily embedded in a research paper.

There are other parts of the Joshua pipeline that might benefit from visualization. During rule extraction, SCFG rules are automatically generated given an aligned parallel corpus. Being able to visually inspect both the aligner output and the results of the rule extraction on an individual phrase could provide some insight into this process. The decoder works by CKY parsing, so it would be advantageous for researchers to be able to view the parse chart that is produced — which constituents are generates where, what their associated weights are, which ones have been pruned, and so

on. This would help to determine if translation errors are caused by search errors or something else.

Acknowledgments

This research was supported in part by the EuroMatrixPlus project funded by the European Commission under the Seventh Framework Programme, and by the US National Science Foundation under grant IIS-0713448. The views and findings are the authors' alone.

Bibliography

Albrecht, Joshua, Rebecca Hwa, and G. Elisabeta Marai. Correcting automatic translations through collaborations between MT and monolingual target-language users. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 60–68, Athens, Greece, March 2009. URL http://www.aclweb.org/anthology/E09-1008.

Billott, Sylvie and Bernard Lang. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June 1989. URL http://www.aclweb.org/anthology/P89-1018.

Chiang, David. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-2005)*, Ann Arbor, Michigan, 2005.

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

DeNeefe, Steve, Kevin Knight, and Hayward H. Chan. Interactively exploring a machine translation model. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 97–100, Ann Arbor, Michigan, June 2005. doi: 10.3115/1225753.1225778. URL http://www.aclweb.org/anthology/P05-3025.

Klein, Dan and Chris Manning. Parsing and hypergraphs. In *In Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2001. URL http://www.cs.berkeley.edu/~klein/papers/klein_and_manning-parsing_and_hypergraphs-IWPT_2001.pdf.

Li, Zhifei, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, March 2009. URL http://www.aclweb.org/anthology/W/W09/W09-0x24.

Zollmann, Andreas and Ashish Venugopal. Syntax augmented machine translation via chart parsing. In *Proceedings of the NAACL-2006 Workshop on Statistcal Machine Translation (WMT-06)*, New York, New York, 2006.