The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010

EDITORIAL BOARD

Published twice a year by Charles University in Prague

Editorial office and subscription inquiries:
ÚFAL MFF UK, Malostranské náměstí 25, 118 00, Prague 1, Czech Republic
E-mail: pbml@ufal.mff.cuni.cz

ISBN 978-80-904175-4-0
ISSN 0032-6585

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010

CONTENTS

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010

EDITORIAL

Special Issue on Open Source Machine Translation Tools

For the second time, we are able to present a special issue on open source tools in machine translation. As part of the Machine Translation Marathon, held on 25–30 January 2010 in Dublin, an Open Source Convention brought together a diverse group of researchers who made practical implementations of building blocks for machine translation systems available as open source software. This software is described in the following papers.

Annually since 2007, the Machine Translation Marathon events have been organized by the EuroMatrix and EuroMatrixPlus projects, which are funded by the European Commission (Framework Programme 6 and 7). The events attract around 50–100 attendees. In its fourth installment, it consists of a winter school, research talks, and the open source convention that is reflected in this special issue.

In part, this special issue builds on the success of the MT Marathon held in the previous year in Prague, where a similar event attracted nine papers. This year we received 19 submissions which were reviewed by a program committee. 16 papers were accepted for presentation at the Marathon and publication in this special issue.

In a larger part, however, this special issue is a testament to the new world of interconnected research that goes beyond the confinements of individual labs, and enables global collaboration — thanks to the technological advances of computer networks and due to the willingness to freely share the fruits of the hard work. Researchers in the field increasingly recognize the advantages of their methods being used (and their work being cited), and the shared benefits of collaborative research.

Phillipp Koehn
Co-editor of the special issue
pkoehn@inf.ed.ac.uk

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 7–16

A Productivity Test of Statistical Machine Translation Post-Editing in a Typical Localisation Context

Mirko Plitt, François Masselot

Autodesk Development Sàrl, Neuchâtel, Switzerland

Abstract

We evaluated the productivity increase of statistical MT post-editing as compared to traditional translation in a two-day test involving twelve participants translating from English to French, Italian, German, and Spanish. The test setup followed an empirical methodology. A random subset of the entire new content produced in our company during a given year was translated with statistical MT engines trained on data from the previous year. The translation environment recorded translation and post-editing times for each sentence. The results show a productivity increase for each participant, with significant variance across inviduals.

1. Introduction

The machine translation productivity test described in this article was conducted in the context of the deployment of machine translation at Autodesk, a software company whose products are translated ("localised") from English into up to twenty languages. We held this test to manage expectations as to the financial savings our company would be able to achieve thanks to machine translation.

Publicly available data on post-editing productivity of statistical machine translation in localisation is scarce (O´Brien, 2005; Takako et al., 2007; Schmidtke, 2008; De Sutter et al., 2008; Flournoy and Duran, 2009). Furthermore, most of the data that is available has not been acquired under controlled conditions (Krings, 2001).

Specific limitations of other post-editing productivity tests that prevented us from using their results included:

- Unclear test objectives leading e.g. to non-representative training corpora.
- Untypical translator profiles.

Corresponding author: mirko.plitt@autodesk.com
Cite as: Mirko Plitt, François Masselot. A Productivity Test of Statistical Machine Translation Post-Editing in a Typical Localisation Context. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 7–16. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0010-x.

- Artificial test sets (e.g. because of a close relation with the corpus[1], or because text deemed unsuitable for MT was removed);
- Absence of traditional translation benchmarks (e.g. assuming a daily throughput of 2500 words, a common rule of thumb in the localisation industry).
- Unreliable time measurement (e.g. based on times reported by individual participants), if any.
- Commercial bias.

The machine translation system we selected for our productivity test was the open-source Moses system (Koehn et al., 2007), trained solely on our own data without any factored representation.

We chose Moses for the following main reasons: (i) the language-independent nature of statistical machine translation makes it easily expandable across several languages at once; (ii) as a typical translation service buyer we possess considerable amounts of high-quality legacy translations; (iii) it would have been difficult to reach return on investment with a commercial machine translation system.

2. Test Setup

The principal aim of our productivity test was to measure the productivity increase we could expect in production at Autodesk. The actual productivity numbers presented in this article may therefore be of limited use for other users of machine translation. Elements of the experimental approach we took to obtain these numbers, however, can be applied beyond our specific case. The following aspects of our approach merit particular attention:

2.1. Test Set Selection

We simulated a Moses production deployment in the most recent round of translation.[2] We therefore trained engines on all our translation data up to the end of 2008. The test set was a randomly selected subset of all the new[3] content submitted for translation in *2009*.

The random selection ensured that the test set was representative in every sense, including any phenomenon that may or may not influence MT quality and post-editing productivity. We split the test set into "jobs", grouped by product (to preserve some context), and sentences were kept in their original order (if often separated by gaps).

[1] We believe that the practice of "cutting" a test set from a corpus presents the risk of introducing a bias in the relation between the two.

[2] The majority of Autodesk products are released once per year; translation activity therefore follows yearly cycles.

[3] *New* means, in this instance, sentences yielding translation memory matches below 75%. In a typical localisation scenario, the use of translation memory technology leaves little room for the deployment of MT above this threshold (Bruckner and Plitt, 2001; Carl and Hansen, 1999).

2.2. Post-Editing Environment

To measure translation time as precisely as possible, without relying solely on what test participants would track and report back to us, we developed our own "workbench", a post-editing environment largely inspired by the caitra environment (Koehn and Haddow, 2009). The workbench was designed to capture keyboard and pause times for each sentence, and was implemented in Ruby on Rails, a web application framework that readily offered most of the functionality required.

The workbench interface (see Figure 1) presented the source and target sentences one beneath the other. For the post-editing tasks, the target sentence field was prepopulated with the MT proposal, to prevent test participants from translating from scratch. The workbench recorded the edit time, the number of edit sessions and the number of key strokes for each sentence.

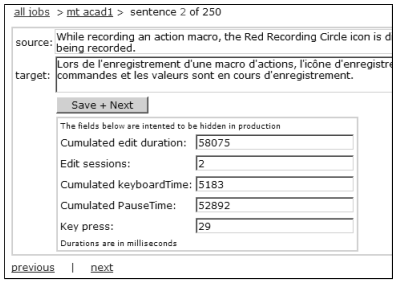

Figure 1. Workbench screenshot (time recording fields were hidden from translators)

2.3. Test Participants

We chose three of our usual localisation vendors for the test. Each vendor assigned one translator per language. We did not intervene in the translator selection as such, and did not request candidates to present particular profiles in terms of translation speed or quality, or post-editing experience. We did not provide our test participants with any training but gave simple post-editing instructions.

2.4. Translation Productivity Benchmark

The productivity test was divided in two phases; the first phase consisted of traditional translation without support from MT—to obtain a reference value for each individual test participant—and only the second phase was dedicated to post-editing. We assigned the jobs in such a way that each translator was to do at least one job in

each of the three product domains, both in post-editing and translation. We also made sure that participants would not translate and post-edit the same job.

2.5. Quality Assessment

Our expectation was that the quality of the post-edited translation would be equivalent to traditional translation, quality being defined here according to the standard criteria applied at Autodesk.

To verify that this expectation was met, we provided the Autodesk translation QA team with samples of translated and post-edited text, again randomly selected, and of reasonable size. The QA team was aware of the overall context of the productivity test but did not know which text was the result of post-editing and which was a traditional translation.

2.6. Test Execution

The test was scheduled to last two days. The source language was English, and the target languages were French, Italian, German, and Spanish. Given that we had opted for three translators per language, there were a total of twelve test participants.[4] The scope of the test was defined by what we considered the minimum of meaningful data at a reasonable cost compared to the anticipated savings potential in production.

We prepared 96 jobs, of which 75 ended up being processed, some entirely, some only partially. The cross-product of jobs, languages, and translation types corresponds to 144,648 source words processed.

A small number of sentences, 1.6%, had a duration above five minutes and up to three hours, cumulating to a total 22% of the the time recorded, without there being any explanation such as the complexity of the source text. These sentences were removed from the result set.

3. Test Results

3.1. Throughput

Figure 2 summarises the test results in terms of throughput. It is most interesting to look at the throughput delta between translation and post-editing for a given translator. Absolute throughputs range from 400 to 1800 words per hour.

Variance across translators was high. MT allowed all translators to work faster, though in varying proportions: from 20% to 131%[5]. MT allowed translators to improve their throughput on average by 74%; in other words, MT saved 43% of the translation time.[6]

[4] One translator chose not to correct tag positions in MT proposals. This translator's work was discarded.

[5] where a 100% productivity gain corresponds to doubling the throughput.

[6] $1 - \frac{1}{1+0.74} = 0.43$

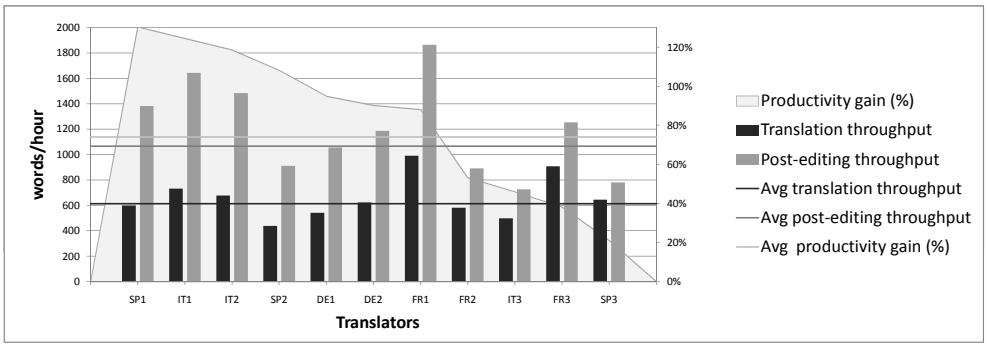

Figure 2. Individual productivity in words per hour (sorted by descending productivity gain)

Figure 3 illustrates that in our test, the benefits from MT were greater for slower than for faster translators. Fast translators presumably have a smaller margin of progression because they have already optimised their way of working.

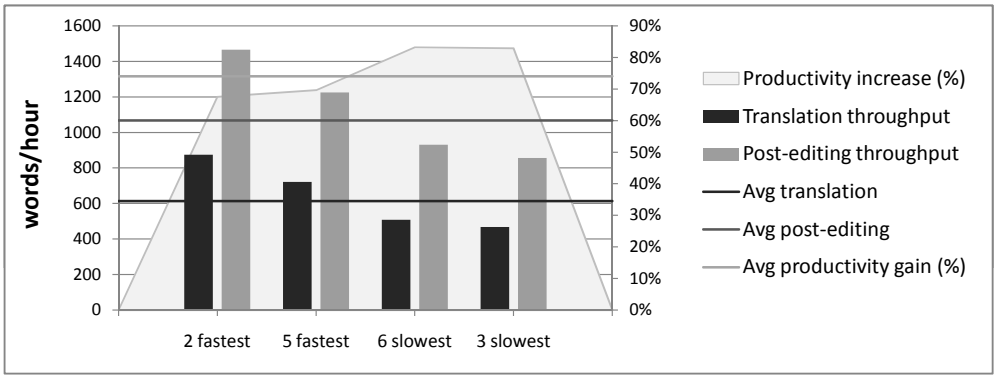

Figure 3. Fast and slow translators

3.2. Edit Distance and Post-Editing Effort

We calculated edit distances to measure the post-editing effort. We used four different scoring methods: Non-Edited, (sentence-level) BLEU (Papineni et al., 2002), Word Error Rate (WER) (Hunt, 1989; McCowan et al., 2005) and Position-independent

Error Rate (PER) (Tillmann and Ney, 2003). Non-Edited represents the ratio of sentences that were left unchanged. We found that these four indicators, despite their different computing methods, correlate relatively well.

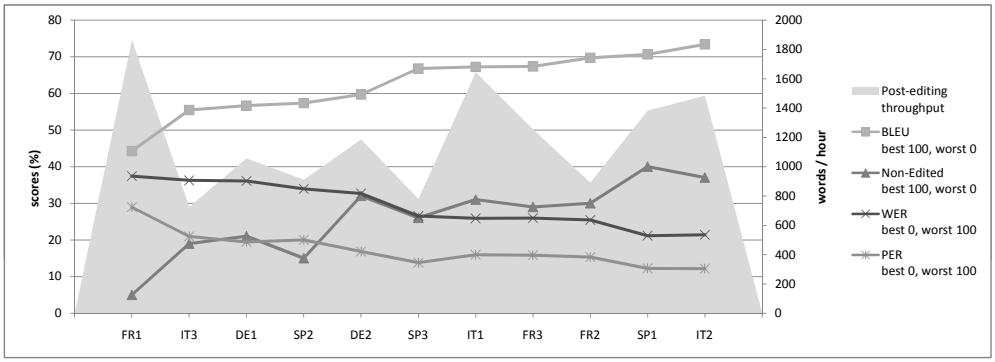

Figure 4. Post-editing throughput and edit distance (sorted by ascending BLEU score)

Figure 4 shows a comparison between post-editing throughput and edit distance. One could intuitively expect that fast translators make fewer changes than slow translators. In our test, however, the post-editor who made the highest number of changes was also the fastest. The graphs indicate no clear correlation between edit distance and throughput.

3.3. Sentence Length

We also examined the relation between the time spent on sentences and the number of words they contained. Figure 5 shows linear regression for segments up to 35 words.[7]

Figure 6 shows the throughput in *words per hour*, in relation to sentence length. An optimum throughput appears to be reached for sentences of around 25 words. Optima for translation and post-editing are relatively close: around 25 words for translation and 22 words for post-editing. The shapes of the polynomial regression curves indicate that the negative impact of very long sentences on throughput is greater for post-editing.

The productivity gain from post-editing corresponds to the vertical distance between the lines; the optimum is situated around 22 words per sentence. 20–25 word sentences are probably more likely to be semantically self-contained than shorter sentences, thus requiring fewer context checks. The minimal time spent on the translation

[7]Sentences with more than 35 words were infrequent in our test set.

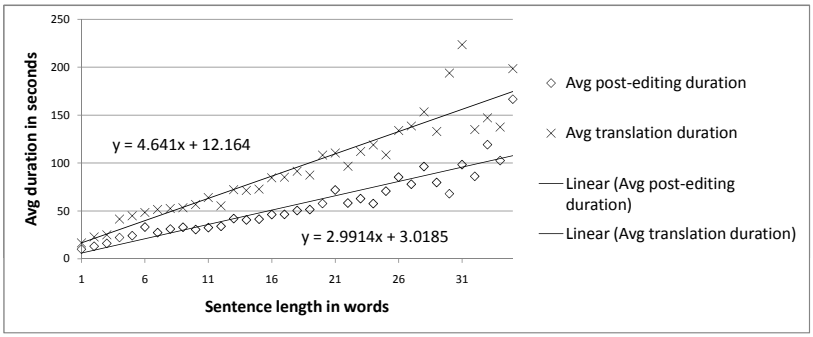

Figure 5. Average duration and sentence length

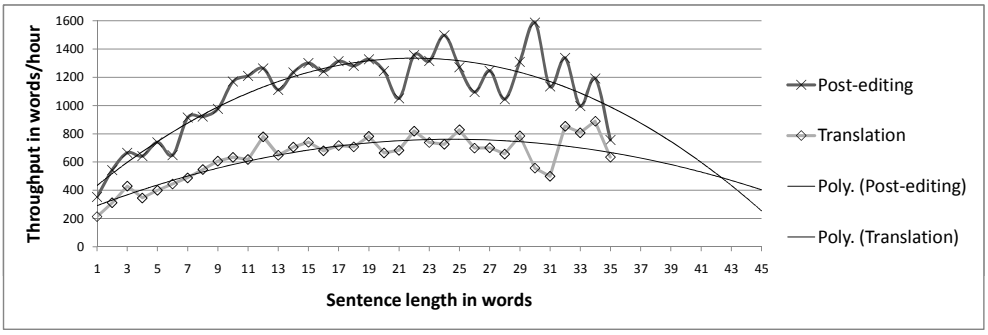

Figure 6. Words per hour and sentence length

of any sentence, including the navigation within the text, plays also a proportionally bigger role for shorter sentences.

3.4. Influence of Language and Product Domain

Average throughput of translators by language essentially reflects individual differences. Our data does not suggest that MT is more suited for one of the four test languages than for another. We were surprised that the productivity increase of German translators was in line with their French, Italian, and Spanish colleagues, despite the lower quality of the German output that we perceive ourselves.

There was no indication either that the content taken from one product was more suitable, or less, for post-editing than content from other products.

3.5. Keyboard Time versus Pause Time

We only recorded two types of editing time: keyboard time and pause time[8]. Pause time can be assumed to include activities such as reading, thinking, and consulting of references.

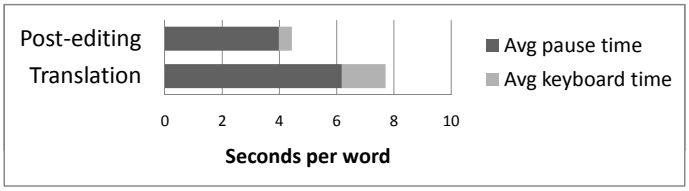

Figure 7. Keyboard and pause time per word

Figure 7 shows that keyboard time represents 19% of the edit time for translation and only 10% for post-editing. MT reduces keyboard time by 70% and pause time by 31%. It seems logical that a good MT proposal saves typing time, but it also saves a third of the "thinking" time.

Keyboard and pause time variations were consistent across products, languages and individuals.

3.6. Work Regularity

Figure 8 plots, for each job, the standard deviation of the seconds-per-word data series recorded for each sentence. The data suggests that MT evens out the work pace of translators. Our interpretation of this result is that the positive impact of the presence of MT proposals is not only limited to a subset of content or to specific types of sentences.

3.7. Quality Assessment

The Autodesk linguistic quality assurance team reviewed part of the jobs of ten of the twelve test participants, evenly split between translation and post-editing jobs for each language. The team rated all the jobs reviewed as either average or good, so all would have been published as is.

The proportion of sentences for which our QA team flagged corrections is grouped in Figure 9. To our surprise, translation jobs contained a higher number of mistakes than post-editing jobs.

[8]keyboard time = sum of time intervals separating two key strokes *inferior* to one second; pause time = sum of time intervals separating two key strokes *superior* to one second

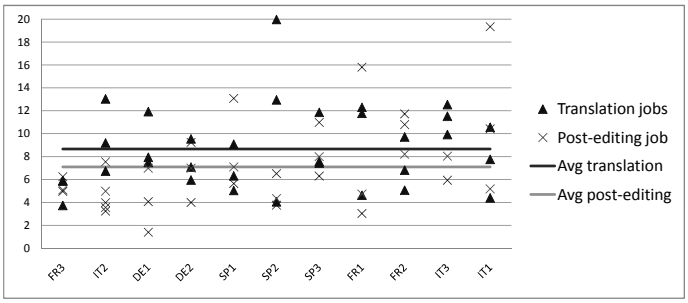

Figure 8. Standard deviation of seconds per word for each job

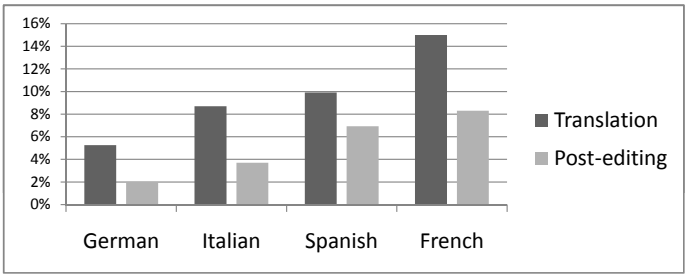

Figure 9. Percentage of sentences with translation errors

3.8. Translator Feedback

The test participants sent us ample feedback on their experience. On the whole, their comments matched our observations and showed that the test had worked well from their perspective too. However, some of the attempts to interpret their experience were in contradiction with our observations, such as an alleged loss of productivity on longer sentences. There also was contradictory feedback from different participants related to the correctness of product terminology.

4. Conclusion

Our test showed that the post-editing of statistical machine translation, when trained and used on Autodesk data, allows translators to substantially increase their productivity. Autodesk has since deployed Moses in production. The empirical methodology followed in the test setup and described in this article can be applied to other real-world evaluations of post-editing productivity.

Bibliography

Bruckner, Christine and Mirko Plitt. Evaluating the operational benefit of using machine translation output as translation memory input. In *MT Summit VIII, MT evaluation: who did what to whom (Fourth ISLE workshop)*, pages 61–65, Santiago de Compostela, Spain, 2001.

Carl, Michael and Silvia Hansen. Linking translation memories with example-based machine translation. In *Machine Tranlation Summit VII*, pages 617–624, Singapore, Singapore, 1999.

De Sutter, Nathalie, Marie-Laure Poëte, and Joeri Van de Walle. Machine translation productivity evaluation report. Unpublished report on the evaluation of two commercial MT systems conducted for Autodesk, April 2008.

Flournoy, Raymond and Christine Duran. Machine translation and document localization at adobe: From pilot to production. In *MT Summit XII: proceedings of the twelfth Machine Translation Summit*, Ottawa, Ontario, Canada, August 2009.

Hunt, Melvyn J. Figures of merit for assessing connected-word recognisers. In *SIOA-1989*, volume 2, pages 127–131, 1989.

Koehn, Philipp and Barry Haddow. Interactive Assistance to Human Translators using Statistical Machine Translation Methods. In *MT Summit XII*, 2009.

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. Moses: Open source toolkit for statistical machine translation. In *ACL Companion Volume. Proc. of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Krings, Hans Peter. *Repairing texts: empirical investigations of machine translation post-editing processes*. Kent State University Press, Kent, Ohio, USA, 2001.

McCowan, Iain, Darren Moore, John Dines, Daniel Gatica-Perez, Mike Flynn, Pierre Wellner, and Hervé Bourlard. On the use of information retrieval measures for speech recognition evaluation. Technical report, Idiap Research Institute, Martigny, Switzerland, March 2005.

O´Brien, Sharon. Methodologies for measuring the correlations between post-editing effort and machine translatability. *Machine Translation*, 19(1):37–58, March 2005.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

Schmidtke, Dag. Microsoft office localization: use of language and translation technology. 2008. URL `http://www.tm-europe.org/files/resources/TM-Europe2008-Dag-Schmidtke-Microsoft.pdf`.

Takako, Aikawa, Lee Schwartz, Ronit King, Mo Corston-Oliver, and Carmen Lozano. Impact of controlled language on translation quality and post-editing in a statistical machine translation environment. In *Proceedings of the MT Summit XI*, Copenhange, Denmark, October 2007.

Tillmann, Christoph and Hermann Ney. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Comput. Linguist.*, 29(1):97–133, 2003.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 17–26

Sulis: An Open Source Transfer Decoder for Deep Syntactic Statistical Machine Translation

Yvette Graham

National Centre for Language Technology, Dublin City University, Ireland

Abstract

In this paper, we describe an open source transfer decoder for Deep Syntactic Transfer-Based Statistical Machine Translation. Transfer decoding involves the application of transfer rules to a SL structure. The N-best TL structures are found via a beam search of TL hypothesis structures which are ranked via a log-linear combination of feature scores, such as translation model and dependency-based language model.

1. Introduction

Deep Syntactic Transfer-Based Statistical Machine Translation (SMT) applies standard methods of Phrase-Based SMT (PB-SMT) (Koehn et al., 2003) to transfer between deep syntactic structures. For example, both Riezler and Maxwell (2006) and Bojar and Hajič (2008) use beam search decoding and a log-linear model to combine feature scores when transferring from source language (SL) deep syntactic structure to target language (TL) deep syntactic structure. Each uses different statistical models for transfer, however. Bojar and Hajič (2008) use the Synchronous Tree Substitution Grammar formalism, in which the probability of attaching pairs of dependency treelets into aligned pairs of frontiers given frontier state labels is used as a main feature function, as well as a bigram dependency-based language model. Riezler and Maxwell (2006), on the other hand, use a translation model computed from relative frequencies of automatically induced transfer rules and a trigram dependency-based language model. Riezler and Maxwell (2006) diverge somewhat from PB-SMT, however, by only applying the language model after decoding on the n-best decoder

 Corresponding author: ygraham@computing.dcu.ie
Cite as: Yvette Graham. Sulis: An Open Source Transfer Decoder for Deep Syntactic Statistical Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 17–26. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0005-7.

output.[1] In this paper, we describe the Sulis transfer decoder that, like Riezler and Maxwell (2006), uses a translation model computed from relative frequencies of automatically induced transfer rules, but for language modeling, like Bojar and Hajič (2008), we use a dependency-based language model during transfer decoding to help decide the n-best TL structures. In addition, we increase the dependency-based language model from a bigram model, as in Bojar and Hajič (2008), to a trigram model.

Sulis forms part of a larger system consisting of several resources, most of which are open source and all of which are at least available to the research community: XLE (Kaplan et al., 2002) for parsing and generation, Giza++ (Och et al., 2000) and Moses (Koehn and Hoang, 2007) for automatic word alignment, RIA (Graham and van Genabith, 2009) for automatic transfer rule induction, Ariadne and SRILM (Stolcke, 2002) for dependency-based language modeling and ZMERT (Zaidan, 2009) for Minimum Error Rate Training (Och, 2003) (MERT). Graham et al. (2009) contains the most recently published evaluation of Sulis.

2. Deep Syntactic Transfer-Based SMT

Deep Syntactic Transfer-Based SMT is composed of three parts, (i) parsing to deep syntactic structure, (ii) transfer from SL structure to TL structure and (iii) generation of TL sentence. Step (ii) involves a statistical search for the n-best TL deep syntactic structures by means of a transfer decoder that constructs TL structures by applying transfer rules to the SL structure. Transfer rules are similar to phrases in PB-SMT: a bitext corpus is firstly word-aligned before all rules consistent with the word-alignment are extracted. They differ from SMT phrases, however, in their structure: they are in the form of dependency graphs with missing arguments replaced by variables as opposed to linear sequences of words. As in PB-SMT, for each SL input structure their exists a large number of possible TL output structures. We use beam search to manage the large search space. TL hypotheses are ranked using a log-linear model to combine several feature scores, such as dependency-based language model and translation model. MERT is carried out on a development set to optimize the weights used to combine feature scores.

2.1. Decoding

2.2. Translation Model

As in PB-SMT, a Transfer-Based SMT translation model can be defined as a combination of several feature functions combined using a log-linear model:

$$p(e|f) = exp \sum_{i=1}^{n} \lambda_i h_i(e, f)$$

[1] Through personal communication with John Maxwell.

2.2.1. Transfer Rule Probabilities

In PB-SMT the translation of an input sentence into an output sentence is modeled by breaking down the translation of the sentence into the translation of a set of phrases. Similarly, for Transfer-Based SMT, the transfer of the SL structure **f** into a TL structure **e** can be broken down into the transfer of a set of rules $\{\bar{f}, \bar{e}\}$:

$$p(\bar{f}_1^I|\bar{e}_1^I) = \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)$$

We compute all rules from the training corpus and estimate the translation probability distribution by relative frequency of the rules:

$$\phi(\bar{f}, \bar{e}) = \frac{count(\bar{e}, \bar{f})}{\sum_{\bar{f}_i} count(\bar{e}, \bar{f}_i)}$$

This is carried out in both the source-to-target and target-to-source directions.

2.2.2. Lexical Weighting

In PB-SMT, lexical weighting is used as a back-off since it provides richer statistics and more reliable probability estimates. Adapting this feature to deep syntax is straightforward. In PB-SMT the lexical translation probability of a phrase pair is calculated based on the alignment between the words in the phrase pair. For deep syntactic transfer, we simply calculate the same probability via the alignment of lexical items in the LHS and RHS of a transfer rule. The lexical translation probability of a RHS, $\bar{e}$, given the LHS, $\bar{f}$ and alignment a, is estimated as follows:

$$lex(\bar{e}|\bar{f}, a) = \prod_{i=1}^{length(\bar{e})} \frac{1}{|\{j|(i,j) \in a\}|} \sum_{\forall(i,j)\in a} w(e_i|f_j)$$

We use lexical weighting in both language directions.

2.2.3. Transfer Rule Application

Decoding takes a single SL structure as input and involves a statistical search for the n-best TL structures. The current decoding algorithm works by creating TL solutions via a top-down application of transfer rules to the SL structure beginning at the root.[2] When the LHS of a rule unifies with the SL structure, the RHS produces a portion of TL structure. Figure 1 shows an example application of three rules to the dependency structure for the German sentence *Die Katze schläft gern* 'The cat likes to sleep' shown in Figure 1(a). Figure 1(b) shows the first transfer rule applied to the root node of the SL structure producing the TL structure portion shown in Figure 1(c). Transfer rule variables map arguments in the SL structure to the desired position

[2] In future work, we plan to extend the decoder by allowing rule application starting at any node in the SL structure.

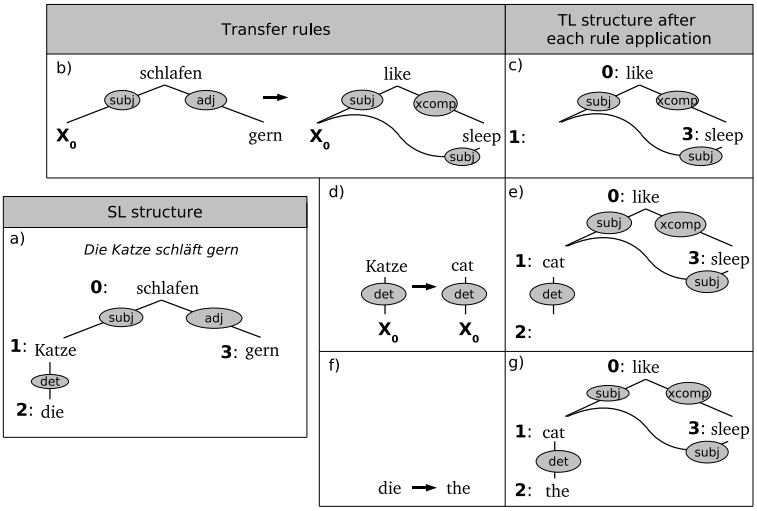

Figure 1. Example top-down application of transfer rules

when creating a TL solution. For example, variable X_0 in Figure 1(b) maps the *subject* of *schlafen* to the *subject* of *like* in the TL structure labeled with id number *1* shown in Figure 1(c). Next *Katze* in the SL structure is translated (Figures 1(d) and 1(e)), before finally *die* is translated (Figures 1(f) and 1(g)).

2.2.4. Beam Search

Partial translations (or translation hypotheses) are constructed by applying transfer rules to the SL structure. While TL translations are constructed, beam search manages the large search space by ranking translation hypotheses and pruning the search by dropping lower scoring hypotheses. A number of stacks are used to organize translation hypotheses into groups of comparable hypotheses, according to the portion of SL structure that has already been translated to produce each hypothesis, i.e. hypothesis stack *N* stores TL translation hypotheses with *N* nodes covered in the SL structure. For example, Figure 2(a) shows the hypothesis stacks for decoding the dependency structure of *Die Katze schläft gern* containing 4 nodes and therefore requiring stacks 1-4 for decoding, each stack storing translation hypotheses for solutions covering one to four nodes, respectively.

Transfer rules are indexed by root node so that they can be retrieved quickly to translate SL structure nodes. For example, in Figure 2(a) the rules rooted at node *Katze* are stored together. Since rules are applied top-down to the SL structure (see

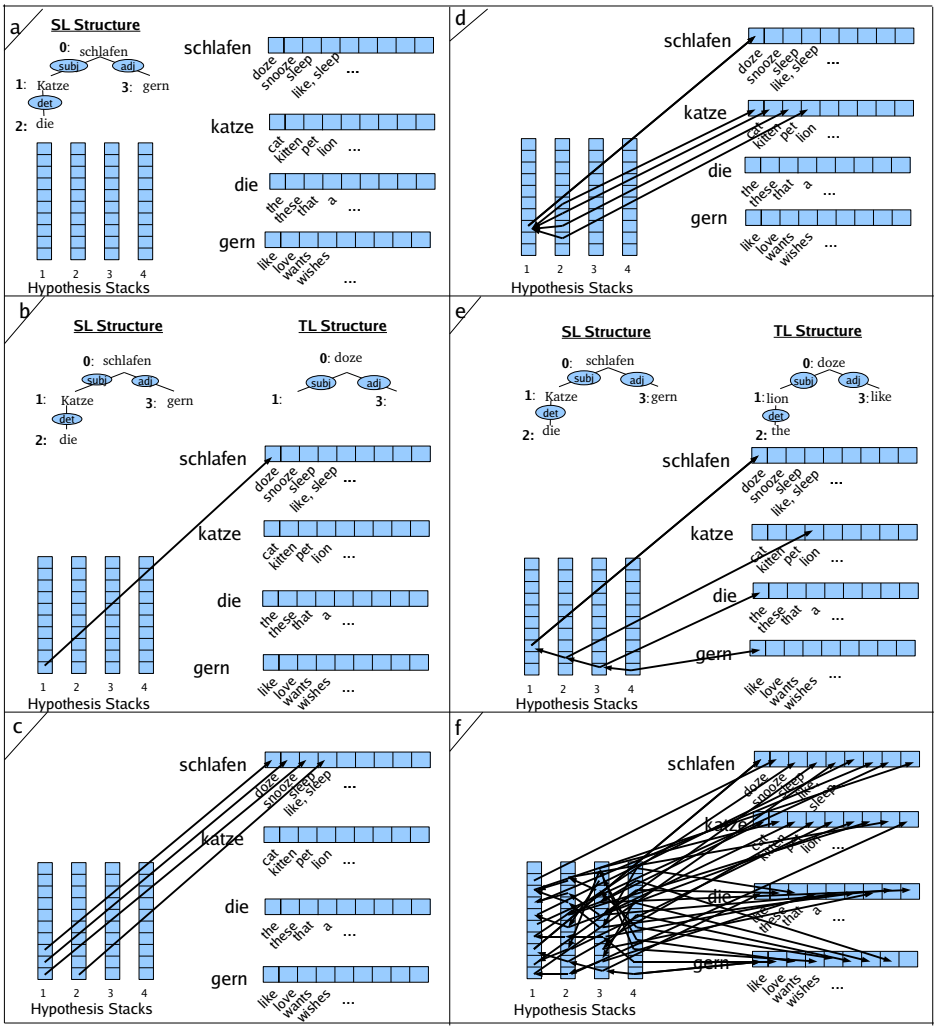

Figure 2. Beam Search Decoding of Example German Deep Syntactic Structure

Section 2.2.3) rules beginning at the root node of the SL structure are first used to construct hypotheses. For example, in Figure 2(b) the rule that translates the root node of the SL structure *schlafen* as *doze* is first used to construct a hypothesis and since it covers one SL node it is stored in hypothesis stack 1. Figure 2(c) shows the next three hypotheses that are constructed: *snooze*, *sleep* and *like sleep*. Hypotheses are ordered within each stack according to their score, high-to-low from bottom-to-top. We currently use histogram pruning. When a stack becomes full, lower scoring solutions are pruned by being popped off the top of the stack.

For efficiency, each partial translation is only stored once in memory even though it may be part of several different future hypotheses. For example, hypothesis stack 2 in Figure 2(d) contains four translations constructed by expanding hypothesis *doze* by four different rules, each translating the word *Katze* into a different TL word. These new hypotheses are represented by a reference to the most recently applied transfer rule (rules translating *Katze*) and a reference back to the previous hypothesis.

Figure 2(e) shows an example of how per single completed translation, the structure for *the lion likes to doze*, is represented in the hypothesis stacks and Figure 2(f) shows all hypotheses represented when the decoder has completed translating a single SL input structure. The n-best translated structures can be retrieved from the final stack.

2.2.5. Efficient Dependency-Based Language Modeling

Although the search space is limited by beam search, during decoding large numbers of TL hypothesis structures need to be ranked. At each expansion of a translation hypothesis (via joining of an existing hypothesis with a transfer rule) a language model score for the newly created hypothesis needs to be calculated. Since this is carried out very many times per single decoding run, it is vital that the method of calculating this score is highly efficient.

In our system, we pre-compute a dependency-based language model score for each transfer rule prior to beam search. This score is calculated only once for each rule even though a single rule may be part of several translation hypotheses. Then during decoding, when a translation hypothesis is expanded by adding a new rule, the new hypothesis score can be calculated quickly by combining the score of the old hypothesis, the rule score and a score calculated based on the probabilities of n-grams where the old hypothesis and rule join together. The probability of a TL hypothesis, h_n, that was produced by combining hypothesis h_{n-1} and rule r can be calculated as follows:

$$\mathtt{hyp\_score}(h_n) = \mathtt{hyp\_score}(h_{n-1}) * \mathtt{join\_score}(h_{n-1}, r) * \mathtt{rule\_score}(r)$$

Since $\mathtt{hyp\_score}(h_{n-1})$ and $\mathtt{rule\_score}(r)$ are already computed, only $\mathtt{join\_score}(h_{n-1}, r)$ needs to be computed to compute $\mathtt{hyp\_score}(h_n)$.

Figure 3 shows how the language model scores are efficiently calculated when decoding the dependency structure for the German sentence *Die Werbung spiegelt die Vielfalt der britischen Universität wider* 'The advertisement reflects the diversity of the

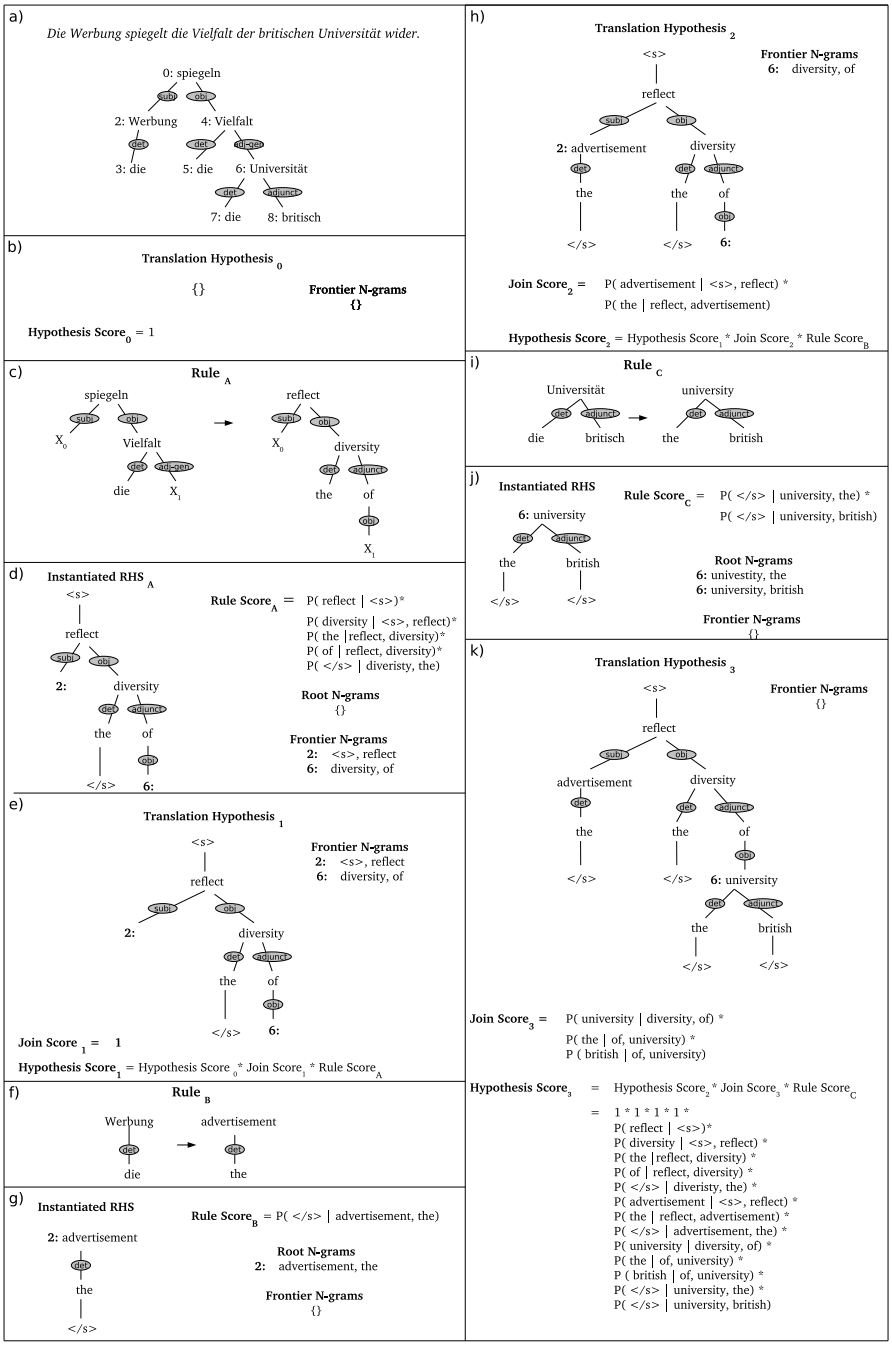

Figure 3. Efficient Dependency-based Language Modeling

British university'. We begin with the German dependency structure graph shown in Figure 3(a) with nodes labeled by id numbers. Figure 3(b) shows the initial empty translation hypothesis that has probability 1.

Figures 3(c), 3(f) and 3(i) show example transfer rules that can be applied to the German dependency structure. Dependency-based language model scores are precomputed for each rule by identifying all trigrams within the RHS structure and calculating the product of their individual probabilities; we call this the rule_score (see Figure 3(d) for Rule_A, Figure 3(g) for Rule_B and Figure 3(j) for Rule_C). In addition, for each rule, n-grams located at the RHS root node and frontier nodes are recorded. For example, Rule_B in Figure 3(g) has a single root node bigram *advertisement the* located at node 2, and Rule_A in Figure 3(d) has two frontier bigrams < *s* >, *reflect* and *diversity, of* located at nodes 2 and 6, respectively. This information is used to calculate the score of joining a rule and a hypothesis.

Figure 3(e) shows the translation hypothesis established by applying Rule_A to the German structure. The language model score for the structure is computed by combining the score of the previous hypothesis (since this is the first rule for this hypothesis, the previous hypothesis is the empty hypothesis and is therefore 1), the join score (since we are joining the rule with the empty hypothesis this score is also 1) and the rule score of Rule_A (see Figure 3(d)).

Figure 3(h) shows the translation hypothesis created by expanding $\mathrm{Hypothesis}_1$ by Rule_B. Since this expansion involved adding a rule at node 2 in the TL structure, the joining trigrams are derived by creating lists of words via all possible combinations of the frontier bigrams belonging to $\mathrm{Hypothesis}_1$ labeled 2 and the root bigrams of Rule_B, also labeled 2 (see root n-grams in Figure 3(g)). For this example, this results in a single word sequence <*s* >*reflect advertisement the* which forms two trigrams <*s* >-*reflect-advertisement* and *reflect-advertisement-the*. The score for $\mathrm{Hypothesis}_2$ is then calculated by combining the hypothesis score for $\mathrm{Hypothesis}_1$, this join score and the precomputed rule score for Rule_B.

3. Using Sulis Decoder

3.1. Input Format

To use Sulis, go to *http://www.computing.dcu.ie/~ygraham/software.html* and follow instructions. The input to the decoder is a text file containing transfer rules and information for computing feature scores. Figure 4 shows an example rule entry for the decoder. The first two lines in Figure 4 gives the id number of the SL structure, 0, and the id number of the node in the SL structure where this rule is applied, also 0. The subsequent lines of the file are used for dependency-based language modeling and are paths of lexical items associated with nodes in the TL structure beginning at the frontier nodes back to the root node of the rule. For example, the rule in Figure 4 has three frontier nodes labeled *1*, *7* and *12*, and therefore the file contains three paths

```
rule: 0
start: 0
1  agree  0  <s>  -1
7  agree  0  <s>  -1
12  agree  0  <s>  -1
cf(1,eq(attr(var(0),'COMP'),var(12)))
cf(1,eq(attr(var(0),'PASSIVE'),-))
cf(1,eq(attr(var(0),'SUBJ'),var(1)))
cf(1,eq(attr(var(0),'PRED'),semform(agree,_17367,[var(1),var(12)],[])))
cf(1,eq(attr(var(0),'TENSE'),pres))
cf(1,eq(attr(var(0),'ADJUNCT'),var(6)))
cf(1,in_set(var(7),var(6)))
lhs_vars: 0
num_rhs: 1
str: 8.204571144249204
tsr: 8.519636252843213
stl: 7.730179751898788
tsl: 8.799261640333976
```

Figure 4. Example Rule Entry of Input File: rule translating German structure word meinen *as* agree*.*

from each of these nodes back to the root.[3] Next is the RHS of the rule.[4] Following that, is a list of SL nodes covered by the LHS of the rule. In the example in Figure 4 a single node, labeled *0* is covered by the rule. In addition, the number of TL lexicalized nodes produced by the rule is given. Finally, the source-to-target and target-to-source relative frequencies are given in the form of positive log probabilities, as well as the source-to-target and target-to-source lexical weights, also as positive log probabilities.

In addition to the rules for each structure, the decoder takes in a weights file for combining feature scores. The file should be in the format of Zaidan (2009) Z-MERT tool. For language modeling, the tool expects a dependency-based language model in ARPA format. To compute such a model, Ariadne open source tool in conjunction with the SRILM toolkit (Stolcke, 2002) can be used.

3.2. Output Format

The decoder outputs the n-best TL structures in the form of the union of the RHS equations of transfer rules used to construct it, as well as a list of feature scores and the total combined score.[5]

[3] Each lexical item in the path is labeled with its node id number, which is used to verify that no single trigram is counted more than once.

[4] In Figure 4, this is in the form of LFG F-structure Prolog equations, but can in fact be in any format, as it is not interpreted by the program code, but simply remains as a string of characters to be output if this rule forms part of a solution.

[5] These scores are needed for MERT.

4. Conclusion

In this paper, we present an open source transfer decoder for Deep Syntactic Transfer-Based SMT. The decoder applies standard methods of PB-SMT to deep syntactic transfer.

Bibliography

Bojar, Ondřej and Jan Hajič. Phrase-Based and Deep Syntactic English-to-Czech Statistical Machine Translation. In *Proceedings of the third Workshop on Statistical Machine Translation*, Columbus, Ohio, June 2008.

Graham, Yvette and Josef van Genabith. An open source rule induction tool for transfer-based smt. *The Prague Bulletin of Mathematical Linguistics Special Issue: Open Source Tools for Machine Translation*, pages 37–46, 2009.

Graham, Yvette, Josef van Genabith, and Anton Bryl. F-structure transfer-based statistical machine translation. In *Proceedings of Lexical Functional Grammar Conference 2009*, Cambridge, July 2009.

Kaplan, Ronald M., Tracy H. King, and John T. Maxwell. Adapting existing grammars: the XLE experience. In *Proceedings of COLING 2002*, Taipei, Taiwan, 2002.

Koehn, Philipp and Hieu Hoang. Factored Translation Models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, page 868–876, Prague, June 2007.

Koehn, Philip, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of HLT-NAACL 2003*, pages 48–54, Edmonton, Alberta, 2003.

Och, Franz Josef. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan, 2003.

Och, Franz Josef, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proceedings of the 1999 Conference on Empirical Methods in Natural Language Processsing (EMNLP 99*, pages 20–28, College Park, MD, 2000.

Riezler, Stefan and John Maxwell. Grammatical Machine Translation. In *Proceedings of HLT-ACL*, pages 248–255, New York, 2006.

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

Zaidan, Omar. Z-mert: A fully configurable open source tool for minimum error rate training of machine translation systems. pages 79–88, 2009.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 27–36

Combining Machine Translation Output with Open Source
The Carnegie Mellon Multi-Engine Machine Translation Scheme

Kenneth Heafield, Alon Lavie

Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213

Abstract

The Carnegie Mellon multi-engine machine translation software merges output from several machine translation systems into a single improved translation. This improvement is significant: in the recent NIST MT09 evaluation, the combined Arabic-English output scored 5.22 BLEU points higher than the best individual system. Concurrent with this paper, we release the source code behind this result consisting of a recombining beam search decoder, the combination search space and features, and several accessories. Here we describe how the released software works and its use.

1. Introduction

Research in machine translation has led to many different translation systems, each with strengths and weaknesses. System combination exploits these differences to obtain improved output. Many approaches to system combination exist; here we discuss an improved version of (Heafield et al., 2009) that, unlike most other approaches, synthesizes new word orderings. Since September 2008, the code we release has been completely rewritten in multithreaded C++ that produces 2.9 combined translations per second. Along with the core system combination code, we also release language modeling and evaluation tools of use to the machine translation community. All of these are available for download at `http://kheafield.com/code/mt/`.

The scheme has several parts. Hypotheses are aligned in pairs using the publicly available METEOR (Banerjee and Lavie, 2005) aligner. A search space (Heafield et al., 2009) is defined on top of these alignments. Beam search is used to make this search tractable. Recombination increases efficiency and diversity by packing hypotheses

Corresponding author: heafield@cs.cmu.edu
Cite as: Kenneth Heafield, Alon Lavie. Combining Machine Translation Output with Open Source: The Carnegie Mellon Multi-Engine Machine Translation Scheme. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 27–36. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0008-4.

that extend in the same way. Hypotheses are scored using a linear model that combines a battery of features detailed in Section 3. Model weights are tuned using Z-MERT (Zaidan, 2009).

The remainder of this paper is organized as follows. Section 2 surveys other system combination techniques. In Section 3 we describe the components of the system with reference to code while Section 4 shows how to run the system. Section 5 summarizes results in recent evaluations and Section 6 concludes.

2. Related Work

Confusion networks (Rosti et al., 2008; Karakos et al., 2008) are a popular form of system combination. This approach combines k-best output from multiple systems. A single k-best list entry is selected as the backbone, which determines word order. The backbone may be selected greedily using some agreement metric or jointly with the full decoding problem (Leusch et al., 2009). Once the backbone is selected, every other k-best entry is aligned to the backbone using exact matches and position information. Translation Edit Rate (Snover et al., 2006) is commonly used for this purpose, with the substitution operation corresponding to alignment. This alignment is still incomplete; unaligned words are aligned to the empty word, corresponding to the deletion (if in the backbone) or insertion (if in a different sentence) operations of TER. Within each alignment, entries vote on word substitution, including with the empty word. Selection of the backbone and word substitution are the only options considered by confusion networks.

The next type of system combination jointly resolves word order and lexical choice. In our approach, we permit the backbone to switch as often as each word. Closely related work (He and Toutanova, 2009) uses a reordering model like Moses (Koehn et al., 2007) to determine word order. While they resolve ambiguous position-based alignments jointly with decoding, we use METEOR to greedily resolve ambiguities resulting from knowledge-based alignments. Since these approaches allow many new word orders, both employ features to control word order by counting n-gram agreement between the system outputs and candidate combination. We use jointly tunable n-gram and system weights for these features; other work uses tunable system weights for at most unigrams (Zhao and He, 2009).

3. Components

3.1. Alignment

Rather than align to a single backbone, we treat single best outputs from each system symmetrically. All pairs are aligned using METEOR. It identifies, in decreasing order of priority:

1. Case-insensitive exact matches

2. Snowball (Porter, 2001) stem matches
3. Shared WordNet (Fellbaum, 1998) synonyms
4. Unigram paraphrases from the TERp (Snover et al., 2008) database

By contrast, confusion networks typically stop with exact matches and use position-based techniques to generate additional alignments. We eschew position-based methods since they can align content words with function words, leading to dropped content not noticed by BLEU (Karakos, 2009). In fact, we replaced the position-based artificial alignments of (Heafield et al., 2009) with the paraphrase database, finding similar performance. The `MEMT/Alignment` directory contains a Java class that calls the publicly available METEOR code to perform pairwise alignments. Since METEOR includes the WordNet database and a tool to extract the paraphrases, neither WordNet nor TERp is required.

3.2. Search Space

The search space is defined on top of the aligned sentences. A hypothesis starts with the first word of some sentence. It can continue to follow that sentence, or can switch to following a different sentence after any word. What results is a hypothesis that weaves together parts of several system outputs. In doing so, we must ensure that pieces cover the sentence without duplication and are fluent across switches. Duplication is prevented by ensuring that a hypothesis contains at most one word from each group of aligned words. A hypothesis may only switch to the first unused word from another output, thereby ensuring that the hypothesis covers the entire sentence. However, this can sometimes be too conservative, so a heuristic permits skipping over words in some cases (Heafield et al., 2009). That paper introduced two choices of heuristic and a radius parameter; here we use the length heuristic with radius 5. Code for the search space appears in the `MEMT/Strategy` directory. We use features to reward fluency.

3.3. Features

Since the search space so easily switches between sentences, maintaining fluency is crucial. A number of features are used for scoring partial and complete hypotheses:

Length The hypothesis length, as in Moses (Koehn et al., 2007). This compensates for the linear impact of length on other features.

Language Model Log probability from a Suffix Array (Zhang and Vogel, 2006) or an ARPA format language model. These appear in the `lm` directory with a simple common interface. We avoid direct dependence on the SRI (Stolcke, 2002) toolkit by providing our own equivalent implementation of inference.

Backoff Average n-gram length found in the language model. This provides limited tunable control over backoff behavior.

Match For each small n and each system, the number of n-gram matches between the hypothesis and system.

Each feature class has a directory under `MEMT/Feature`. Features have a common interface designed to make adding additional features easy. All features are combined in a *linear* model, which is equivalent to a log-linear model with each feature exponentiated. Model weights, especially for the match features, are heavily dependent on the underlying systems. We therefore provide scripts in `MEMT/scripts/zmert` to tune weights using Z-MERT (Zaidan, 2009). With 20 or more features, the optimization part of each iteration typically takes longer than does decoding.

3.4. Beam Search

Since the search space is exponential in the sentence length, we use beam search with recombination. The beam contains a configurable number of hypotheses of equal length; we typically keep 500 hypotheses. In order to increase beam diversity and speed decoding, we recombine hypotheses that will extend in the same way and score proportionally. Hypotheses to recombine are detected by hashing the search space state, feature state, and hypothesis history up to a length requested by the features. Recombined hypotheses are packed into a single hypothesis that maintains pointers to the packed hypotheses. At the end of the sentence, these packed hypotheses comprise a lattice where each node is labeled with the maximum-score path back to the beginning of the sentence. This enables efficient k-best extraction. The beam search decoder is factored into `MEMT/Decoder`. It only knows about the search space and features via template arguments and, therefore, may be independently useful for other left-to-right beam search problems.

4. Running Combination

4.1. Requirements

We assume a UNIX environment with a C++ compiler, Java, and Python. Scripts are provided in `install/` to install Boost, Boost Jam, ICU, and Ruby without requiring root access. Compiling consists of running `bjam release` in the `MEMT` directory. See the `README` file for more information.

A separate tuning set is required to learn parameter weights. This should be held out from system training or tuning data. We recommend reserving at least 400 segments for this purpose. A language model is also required; many use the SRILM toolkit (Stolcke, 2002) to produce ARPA files for this purpose. It should be tokenized the same way as the system outputs. A tokenizer is not provided; one can be downloaded from `http://www.statmt.org/wmt09/scripts.tgz` (Callison-Burch et al., 2009).

4.2. Alignment

The `MEMT/Alignment/MatcherMEMT.java` class uses the METEOR API to infer alignments. It should be compiled by running `MEMT/Alignment/compile.sh`. This script

will also download and install METEOR if necessary. Tokenized system outputs should be placed in text files with one segment per line. Running alignment is straightforward:

```
$ MEMT/Alignment/match.sh system1.txt system2.txt system3.txt >matched
```

4.3. Optional Language Model Filtering

Optionally, an ARPA language model can be filtered to the sentences being combined. The filter checks that an n-gram's vocabulary is a subset of some segment's vocabulary. This is much more strict than testing against the entire set's vocabulary, where words in an n-gram could be spread across several segments. The reduction in size can be dramatic: filtering a 19 GB ARPA file for the NIST MT09 Informal System Combination task produced a 1.4 GB ARPA file. Since the server will load this model into RAM, filtering greatly decreases hardware requirements. The command to filter to any number of matched files, including those with different sets of systems, is:

```
$ cat matched1 matched2 matched3 | MEMT/dist/FilterLM in.arpa out.arpa
```

The filter is fast: it keeps only the vocabularies in memory and takes about 12 minutes to filter a 19 GB model. This language model filter is also available as a separate package that reads one segment vocabulary per line. While phrase table expansion reduces effectiveness for statistical machine translation systems, we were still able to reduce model size by 36% by filtering to 1797 segments. It can also produce segment-level language model files if desired.

4.4. Decoding Server

The actual decoding algorithm runs inside a server process that accepts TCP connections. This avoids reloading the language model, which typically takes longer than performing thousands of combinations. The server is launched by specifying the language model and port:

```
$ MEMT/scripts/server.sh --lm.type ngram --lm.file lm.arpa --port 2000
```

When loading the language model has finished, it will print "Accepting Connections." Except for the language model and some threading options, configuration is sent by clients. Multiple connections with different configurations work properly. The protocol is highly compressible plain text, especially for k-best lists, so we advise using compressed SSH tunneling if the connection between client and server is slow.

4.5. Configuration

Most of the configuration options are set by clients of the decoding server. Figure 1 shows a configuration file without feature weights, which are added by tuning. Important hyperparameters to tweak are:

`horizon` The suboption `radius` controls how long words linger as described in Section 3.2. The `method` of distance measurement can be `length` or nearby `alignment`,

as described in (Heafield et al., 2009). Generally, a larger window works best in the presence of significant reordering. We recommend starting with `length` and a radius of 5.

`verbatim` Match features are called `verbatim` in the code. Two instances are provided; work on more flexible feature instantiation is planned. The idea behind two instances is that one does lexical voting using exact matches while the other uses all alignments to handle support and word order issues. The `mask` option controls which alignment types will count, including the implicit `self` alignment of words and `boundary` markers. The maximum match length to consider is also a key parameter. The `individual` option determines the maximum match length reported individually for each system. This may lead to too many features, so longer n-gram match counts can be presented on a `collective` basis by summing counts across systems.

`ouptut.nbest` Size of n-best output requested.

`length_normalize` This determines if feature values are divided by length, excepting of course the length feature itself. When disabled, the length feature otherwise acts to subtract the impact of length from other features. Empirically, we find turning off length normalization makes the output score slightly higher and output 1-2% longer.

Authoritative documentation of all options is printed when the server is run without an argument:

```
$ MEMT/scripts/server.sh
```

4.6. Tuning

Tuning requires a directory with three files: `decoder_config_base` containing the configuration file from Section 4.5, `dev.matched` containing the aligned tuning sentences from Section 4.2, and `dev.reference` containing the references (one per line). Multiple references for the same segment appear on consecutive lines. Assuming these files are in `work_dir` and the decoding server is running on port `2000`, the command line is:

```
$ MEMT/scripts/zmert/run.rb 2000 work_dir
```

If the server is running on another machine, it may be specified as `host:port`. This will run Z-MERT to tune the system and produce the file `work_dir/decoder_config` with tuned weights. It also decodes the tuning set with this configuration, placing output in `work_dir/output.1best`. Finally, it scores this tuning output against the provided reference, placing results in `work_dir/output.1best.scores`.

4.7. Decoding and Evaluation

Test data is decoded using the tuned configuration file and test `matched` file:

```
$ MEMT/scripts/simple_decode.rb 2000 decoder_config matched output
```

```
output.nbest = 300
beam_size = 500
length_normalize = false

#Remove words more than 5 behind as measured by length.
horizon.method = length
horizon.radius = 5

#Count exact matches up to length 2 for each system.
score.verbatim0.mask = "self exact boundary"
score.verbatim0.individual = 2
score.verbatim0.collective = 2
#Count non-paraphrase matches up to length 2 for each system.
#For length 3 and 4, sum the match counts across systems.
score.verbatim1.mask = "self exact boundary snowball_stem wn_synonymy"
score.verbatim1.individual = 2
score.verbatim1.collective = 4
```

Figure 1. Sample configuration file before tuning weights.

which creates `output.1best` with one segment per line and `output.nbest` in Moses (Koehn et al., 2007) format.

We provide a script that scores translations with BLEU (Papineni et al., 2002) from `mteval-13a.pl` (Peterson et al., 2009), NIST (Doddington, 2003), TER 0.7.25 (Snover et al., 2006), METEOR 1.0 (Banerjee and Lavie, 2005), unigram precision and recall, and length ratio. The following command generates the file `output.1best.scores` containing these respective scores:

```
$ Utilities/scoring/score.rb --hyp-tok output.1best --refs-laced ref
```

Running with `--print-header` will show column headers. Running without arguments provides the full list of options. This script is also available for download as a separate package.

5. Results

The 2009 Workshop on Machine Translation (WMT) (Callison-Burch et al., 2009) and NIST Open MT evaluations (Peterson et al., 2009) both added tracks specifically to evaluate system combination. We participated in both and now present updated unofficial results in Table 1. Gains on NIST data are surprisingly large–but not unexpected given the results from the evaluation (Peterson et al., 2009). Gains on WMT data depend mostly on the gap between Google and other systems; with a large gap, the effectiveness of system combination is minimal.

Source	System	BLEU	TER	METEOR
NIST Arabic	combo	58.55	36.86	70.76
	top single	*51.88*	*40.54*	*67.74*
NIST Urdu	combo	34.72	55.46	53.37
	top single	*32.88*	*56.20*	*52.24*
WMT Czech	combo	21.98	60.48	46.63
	top single	*21.18*	*59.57*	*46.91*
WMT French	combo	31.56	52.48	54.30
	top single	*31.14*	*51.36*	*54.91*
WMT German	combo	23.88	58.29	48.69
	top single	*21.31*	*60.78*	*56.82*
WMT Hungarian	combo	13.84	71.89	36.70
	top single	*12.75*	*68.35*	*35.43*
WMT Spanish	combo	28.79	53.63	53.51
	top single	*28.69*	*53.38*	*54.20*

Table 1. Unofficial post-evaluation scores on test data from past system combination tasks with the top system by BLEU shown in italics for comparison. The NIST MT09 Arabic-English scores are on unsequestered segments only. For 2009 Workshop on Machine Translation results, the language model is constrained; there was no constrained track for MT09 informal system combination. BLEU is uncased (and therefore not the official NIST MT09 metric), TER is version 0.7.25, and METEOR is version 1.0 with hter parameters.

6. Conclusion

We have released the source code to our system combination scheme. It shows significant improvement on some translation tasks, particularly those with systems close in performance. The software is ready to be downloaded, installed, and run. We hope to receive patches from users. In addition to the core system combination code, the language model filter and evaluation script are available as separate packages of general use to the community.

Acknowledgments

Funding for this work was provided by the DARPA GALE program and a National Science Foundation Graduate Research Fellowship. NIST serves to coordinate the NIST Open MT evaluations in order to support machine translation research and to help advance the state-of-the-art in machine translation technologies. NIST Open MT evaluations are not viewed as a competition, as such results reported by NIST are not to be construed, or represented, as endorsements of any participant's system, or as

official findings on the part of NIST or the U.S. Government. Informal System Combination was an informal, diagnostic MT09 task, offered after the official evaluation period.

Bibliography

Banerjee, Satanjeev and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, 2005.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, and Josh Schroeder. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 1–28, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0401`.

Doddington, George. Automatic Evaluation of Machine Translation Quality Using N-gram Co-Occurrence Statistics. In *Proceedings of Human Language Technology Conference*, 2003.

Fellbaum, Christiane. *WordNet: An Electronic Lexical Database*. MIT Press, 1998. ISBN 978-0-262-06197-1.

He, Xiaodong and Kristina Toutanova. Joint optimization for machine translation system combination. In *EMNLP*, August 2009.

Heafield, Kenneth, Greg Hanneman, and Alon Lavie. Machine translation system combination with flexible word ordering. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 56–60, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0408`.

Karakos, Damianos. The JHU system combination scheme. In *NIST Open Machine Translation Evaluation Workshop*, Ottawa, Canada, September 2009.

Karakos, Damianos, Jason Eisner, Sanjeev Khudanpur, and Markus Dreyer. Machine translation system combination using ITG-based alignments. In *Proceedings ACL-08: HLT, Short Papers (Companion Volume)*, pages 81–84, 2008.

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. Moses: Open source toolkit for statistical machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, June 2007.

Leusch, Gregor, Saša Hasan, Saab Mansour, Matthias Huck, and Hermann Ney. RWTH's system combination for the NIST 2009 MT ISC evaluation. In *NIST Open Machine Translation Evaluation Workshop*, Ottawa, Canada, September 2009.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, PA, July 2002.

Peterson, Kay, Mark Przybocki, and Sébastien Bronsart. NIST 2009 open machine translation evaluation (MT09) official release of results, 2009. http://www.itl.nist.gov/iad/mig/tests/mt/2009/.

Porter, Martin. Snowball: A language for stemming algorithms, 2001. http://snowball.tartarus.org/texts/introduction.html.

Rosti, Antti-Veikko I., Bing Zhang, Spyros Matsoukas, and Richard Schwartz. Incremental hypothesis alignment for building confusion networks with application to machine translation system combination. In *Proceedings Third Workshop on Statistical Machine Translation*, pages 183–186, 2008.

Snover, Matthew, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings Seventh Conference of the Association for Machine Translation in the Americas*, pages 223–231, Cambridge, MA, August 2006.

Snover, Matthew, Nitin Madnani, Bonnie Dorr, and Richard Schwartz. TERp system description. In *Proceedings NIST Metrics MATR 2008*, 2008.

Stolcke, Andreas. SRILM - an extensible language modeling toolkit. In *Proc. ICSLP*, pages 901–904, 2002.

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

Zhang, Ying and Stephan Vogel. Suffix array and its applications in empirical natural language processing. Technical Report CMU-LTI-06-010, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, December 2006.

Zhao, Yong and Xiaodong He. Using n-gram based features for machine translation system combination. In *Proceedings NAACL-HLT*, May 2009.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 37–46

Training Phrase-Based Machine Translation Models on the Cloud

Open Source Machine Translation Toolkit Chaski

Qin Gao, Stephan Vogel

InterACT Lab, Language Technologies Institute, Carnegie Mellon University, 407 S. Craig Street, Pittsburgh, PA 15213, United States

Abstract

In this paper we present an opensource machine translation toolkit Chaski which is capable of training phrase-based machine translation models on Hadoop clusters. The toolkit provides a full training pipeline including distributed word alignment, word clustering and phrase extraction. The toolkit also provides an extended error-tolerance mechanism over standard Hadoop error-tolerance framework. The paper will describe the underlying methodology and the design of the system, together with instructions of how to run the system on Hadoop clusters.

1. Introduction

Statistical machine translation relies heavily on data. As the amount of data become larger, the time spent on model training becomes longer. On large scale tasks such as GALE, which scales up to 10 million sentence pairs and more than 300 million words, training on a single machine with GIZA++ (Och and Ney, 2003) and Moses (Koehn et al., 2007) can take more than one week. By applying multi-thread technology to GIZA++, significant speedup can be achieved when multi-core computers are used. However, even with the latest Multi-thread GIZA++ (MGIZA++) (Gao and Vogel, 2008), training a large scale system still requires 5 to 8 days.

A typical phrase-based machine translation training pipeline consists of three major steps: preparing the corpus, word alignment and phrase extraction/scoring. Among these steps, the most time-consuming ones are word alignment and phrase extraction/scoring. Different stages requires different kinds of resources. Take the Moses toolkit as an example. The first step, data preprocessing for word alignment, typi-

Corresponding author: qing@cs.cmu.edu
Cite as: Qin Gao, Stephan Vogel. Training Phrase-Based Machine Translation Models on the Cloud: Open Source Machine Translation Toolkit Chaski. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 37–46. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0004-8.

cally takes 2 to 3 hours, and most of the time is consumed by word clustering, which scales linearly to vocabulary size and quadratic to number of classes[1]. The next step, word alignment, can be split into two stages, the first is generating co-occurrence table which contains all possible word pairs which appear in the corpus, and the second is training IBM models and outputting alignments. The co-occurrence table generation task consumes a large amount of memory. In our profile, running it on a ten million sentence Chinese-English corpus consumed 20GB memory, which may already be a problem for most commodity machines. Again, when training IBM models with GIZA++, the memory usage is essentially smaller but the CPU time becomes the dominant factor. After the alignment is generated, phrase extraction and scoring suffers from a different problem, the I/O bottleneck. When running on large data, phrase extraction requires writing individual phrases onto disk, and later, during scoring stage the phrases will be sorted two times on source or target phrase so as to estimate feature values of each phrase pair. If the size of extracted phrase pairs fits into memory, then internal sorting can be used, however the size of uncompressed phrase pairs can easily grow to 100 GB, as a consequence the `sort` program needs to write and read temporary files which also adds to the burden of disk I/O. It is the reason why phrase extraction, being a relatively simple process, takes also more than two days to finish on a the Chinese-English corpus described above.

With the rapid development of computer clusters, the computational resource is considered abundant. Among the different parallel frameworks, MapReduce is attracting more and more attention (Dean and Ghemawat, 2008). In this framework, two functions, Mapper and Reducer are defined. The Mapper processes raw input and outputs intermediate key-value pairs. The key-value pairs are then sorted and all pairs with the same key will be fed into a reducer instance. With the opensource Hadoop system[2], one can easily set up an error-tolerant cluster with commodity computers, and commercial services such as Amazon EC2 make it even easier to access large Hadoop clusters at small cost. There has been some work on porting machine translation tools to Hadoop: Dyer et al (Dyer et al., 2008) implemented distributed training for IBM 1 and HMM word alignment models based on Hadoop; Venugopal et al (Venugopal and Zollmann, 2009) built an end-to-end syntactic augmented machine translation system on Hadoop. However, there is still no complete toolkit that can handle the whole phrase-based machine translation training pipeline on clusters. In this work we provide a software package toolkit, which ports the whole machine translation training pipeline onto Hadoop clusters, including:

1. **Distributed word clustering**, as the preprocessing step for word alignment.
2. **Distributed word alignment**, for training IBM model 1 to 4 and HMM model.
3. **Distributed phrase extraction**, to extract phrases and score phrase pairs on the cluster.

[1] In Moses, the default number of classes are 50.

[2] Apache Hadoop, http://hadoop.apache.org/

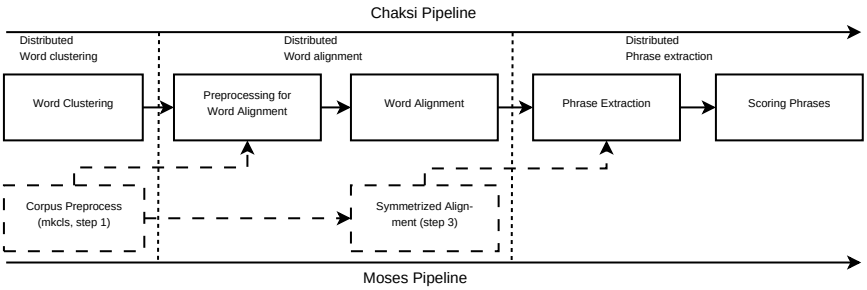

Figure 1. Components in Chaski toolkit and its counter parts in Moses pipeline. The dashed boxes are outputs in Moses pipeline, dashed arrows mean you can take Moses output of the component in Moses and continue training using Chaski.

The final output of the system is a Moses-compatible phrase table and lexicalized reordering model.

In order to handle the training efficiently and reliably on the cluster, the toolkit also takes into account the problem of error-tolerance. MapReduce frameworks such as Hadoop provide primitive error/exception handling mechanism by simply re-running failed jobs. In practice this mechanism does not work well for complex NLP tasks, because exceptions are not necessarily caused by "unexpected" hardware/software problems that can be fixed by restarting. For this kind of exceptions special actions need to be taken so as to recover the training process. In the Chaski system we implemented a cascaded fail-safe mechanism that can apply pre-defined recovery actions to ensure successful training with minimal manual intervention.

In section 2 we will introduce the methodology and implementation of each component in the toolkit. Section 3 provides a brief tutorial of how to setup and run the toolkit. Section 4 presents experimental results on run time and translation, and section 5 concludes the paper.

2. System implementation

The Chaski toolkit consists of three major components, distributed word clustering, distributed word alignment and distributed phrase extraction. Figure 1 shows the pipeline of the system. At the boundaries of each stage, the toolkit is compatible with Moses file formats. In other words, each of the components can be replaced by Moses counter parts. The dashed arrows in Figure 1 demonstrate alternative pipelines. In the remaining part of the section we will first introduce distributed world alignment, then phrase extraction and scoring and finally the cascaded fail-safe mechanism. For distributed word clustering, we re-implemented the algorithm proposed by (Uszkoreit and Brants, 2008), and we refer interested readers to that paper.

2.1. Distributed word alignment

GIZA++ is a widely used word alignment tool. It uses EM algorithm to estimate parameters for IBM models (Brown et al., 1993) and HMM model (Vogel et al., 1996). Given a sentence pair (f_1^J, e_1^I), where f_1^J and e_1^I are source and target sentence with J and I words respectively, an alignment a on the sentence pair is defined as:

$$a \subseteq \mathcal{A}_I^J = \{(j, i) : j = 1, \cdots, J; i \in [0, I]\} \tag{1}$$

in case that $i = 0$ in a link $(j, i) \in a$, it represents that the source word j aligns to an empty target word e_0. In IBM models, the translation probability is defined as the summation of the probabilities of all possible alignments between the sentence pair:

$$P(f_1^J | e_1^I) = \sum_{a \subseteq \mathcal{A}_I^J} P(f_1^J, a | e_1^I) \tag{2}$$

and IBM models consists of several parametric forms of $P(f_1^J | e_1^I) = p_\theta(f_1^J, a_1^J | e_1^I)$. The parameters θ can be estimated by maximum likelihood estimation on training corpus with EM algorithm. The optimal alignment under the current parameter set $\hat{\theta}$ is called Viterbi alignment, as defined in 3, and a large number of state-of-the-art translation systems utilize the Viterbi alignment for phrase or rule extraction.

$$\hat{a}_1^J = \arg\max_{a_1^J} p_{\hat{\theta}}(f_1^J, a_1^J | e_1^J) \tag{3}$$

The algorithm in GIZA++ is an iterative process, and each iteration can be divided into two steps, E-step and M-step. During E-step, the current parameter set $\hat{\theta}$ is used to estimate posteriors of all possible alignments (or a set of n-best alignments for model 3,4 and 5) of all sentence pairs in the training corpus. Then on M-step the posterior of events are summed up and normalized to produce a new parameter set. E-step, which scales linearly to number of sentence pairs, can be time consuming when the size of corpus is large. However, because each sentence pair can be processed independently, it is easy to be parallelized. M-step is relatively fast, however, the step is easily becoming I/O bound in distributed environments if large number of posteriors need to be transferred. In our previous work (Gao and Vogel, 2008), we implemented a multi-thread version of GIZA++ called MGIZA++, and a distributed version, PGIZA++. While MGIZA++ achieved significant speed-up, PGIZA++ suffers from I/O bottleneck in practice. In the new implementation presented in the paper, Hadoop File System (HDFS) is used to collect counts and re-distribute models, and the normalization is implemented as MapReduce tasks, the distributed nature of HDFS greatly improved the efficiency of count collection and re-normalization.

In addition to the I/O bottleneck, when moving towards distributed word alignment, the memory limitation is also a blockage. Hadoop clusters usually limit the memory every process can use, but certain models such as lexical translation model

$p(f_j|e_i)$, is usually too large if no filtering is done. The size of the table is proportional to the source and target vocabulary, hence related to the sizes of chunks of the training corpus. Therefore it is important to estimate the memory footprint and dynamically adjust the sizes of chunks.

The distributed word alignment in Chaski works as follows. First the input corpus is split into chunks. The sizes are dynamically determined by the number of distinct word pairs, which is proportional to the memory footprint. After the chunks are generated, a number of tasks will be started, each handles the E-step of one chunk. The counts are written directly onto HDFS from individual tasks, and the M-step MapReduce tasks are started after all E-step tasks finish. Different M-step MapReduce tasks are implemented for different models with similar ideas that all the counts appearing in the denominator of the normalization formulae will be processed by a same reducer. For example, the counts of lexical translation probability $p(f_j|e_i)$ is a triplet $t = (f_j, e_i, c(f_j|e_i))$, and the normalization formula is $\hat{p}(f_j|e_i) = \frac{\sum_{f=f_j, e=e_i} c(f_j|e_i)}{\sum_{f=f_j} c(f_j|e_i)}$. Therefore we define f_j as the key in Mapper output, so $(f_j, e, c(f_j|e_i)), \forall e$ will go to one reducer, and the reducer has enough information to perform normalization. After normalization is done, the new model will be written to HDFS and the E-step tasks of next iteration will fetch the model and filter it according to the chunk's vocabulary.

2.2. Distributed phrase extraction

Phrase extraction takes the symmetrized word alignments as input and extracts phrases base on pre-defined heuristics. After phrase pairs are extracted, features are assigned to phrase pairs in the scoring phase. Commonly used features include phrase translation probabilities and lexical translation probabilities. Assume a phrase pair (E_i, F_j), where $E = e_1, \cdots, e_K$, $F = f_1, \cdots, f_L$, $e_{1..K}$ and $f_{1..L}$ are words in source/target languages. For phrase translation probabilities, which include two features (source-to-target and target-to-source), the features are defined as:

$$PT_{s\to t}(E_i, F_j) = \frac{\#(E_i, F_j)}{\#(E_i)} \quad (4)$$

$$PT_{t\to s}(E_i, F_j) = \frac{\#(E_i, F_j)}{\#(F_j)} \quad (5)$$

where $\#(E_i, F_j)$ is the count of occurrences of the phrase pair in the corpus, $\#(E_i), \#(F_j)$ are counts of occurrences of source or target phrase in the corpus respectively.

For lexical translation probabilities, which is also bi-directional, we have the definition:

$$LT_{s\to t}(E_i, F_j) = \prod_{k=1}^{K} \left(\frac{\delta|A(e_k)|}{|A(e_k)|} \prod_{f_l \in A(e_k)} p(f_l|e_k) + (1 - \delta|A(e_k)|)p(0|e_k) \right) \quad (6)$$

$$LT_{t\to s}(E_i, F_j) = \prod_{l=1}^{L} \left(\frac{\delta|A(f_l)|}{|A(f_l)|} \prod_{e_k \in A(f_l)} p(e_k|f_l) + (1 - \delta|A(f_l)|)p(0|f_l) \right) \quad (7)$$

where $A(e_k)$ is target word that has alignment with e_k, and $p(0|e_k)$ is the probability of e_k aligned to empty word (not aligned) and $\delta(|A(e_k)|) = 0$ if $A(e_k)$ is empty.

Generally the features can be classified into three categories according to how it can be calculated. For $PT_{s\to t}$, we need all the phrase pairs with a same source phrase, which requires sorting on source phrases, for $PT_{t\to s}$, reversely we need phrase pairs be sorted on target side phrases. Finally the lexical weights can be calculated individually for each phrase pair. Therefore, to get all the four features we need to sort the whole phrase table twice, which can be done in two MapReduce tasks. As shown in Figure 2, the first mapper performs phrase extraction, and output the target phrases as keys, source phrases as values. The MapReduce framework automatically sorts the output on target phrases, and the reducer, which has all the phrase pairs of the same target phrase, can calculate $PT_{t\to s}(E_i, F_j)$. To make the output compact, we do not store all instance of a same phrase pairs, instead we store the phrase pairs with the number of occurrences of the phrase pair. The second mapper works on the output of the first step, the only operation it performs is switching the keys to source phrase, and output both the target phrase pair and the count of the phrase pair. Again the MapReduce framework will sort the output by source phrases, and the reducer can estimate $PT_{s\to t}(E_i, F_j)$. The lexical translation probabilities can be estimated in either reducer, but in implementation we put it on the second reducer. In addition, lexicalized reordering table can be generated within the pipeline, the reordering features are similar to lexical translation probabilities, and is estimated in the second reducer.

2.3. Error-tolerance mechanism

Error-tolerance is an essential part of distributed computing. Hadoop already provides primitive error-tolerance mechanism which is able to re-run failed tasks. However, in many cases, the errors cannot be recovered only by restarting on the same configuration, in such cases the mechanism does not help.

To handle this, we developed a special error tolerance mechanism in Chaski. If error happens, a sequence of actions will be taken to recover the pipeline. The actions will be taken in a cascaded way, first the system will try to re-run tasks on failed data chunks and if it fails for a given number of times, then it will try to reduce the chunk sizes for word alignment or enable disk-based cache for phrase extraction. Finally, if specified by user, the system will try to ignore a certain number of chunks, or stop the pipeline. After user fixed the problem, the pipeline can be resumed from where it stops. This is especially useful for the word alignment step, so that users do not need to restart from beginning.

3. Usage of the software

The toolkit is released under two separated packages, a Java package for Chaski and a C++ package for MGIZA++. Standalone Chaski is capable of distributed word

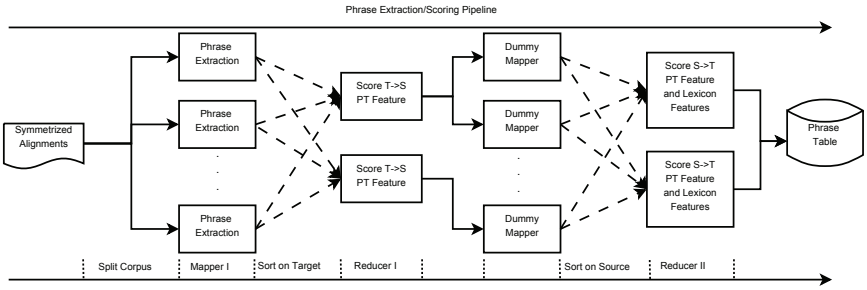

Figure 2. The flowchart of phrase extraction tasks, the dashed arrows represent sorting operations performed by MapReduce framework.

clustering, to perform word alignment MGIZA++ is required. As illustrated in Figure 1, there are multiple alternative pipelines. Chaski can perform full training from raw input data or only perform phrase extraction with the symmetrized alignments generated elsewhere.

After installation of both packages, we need to define two environment variables:

```
$QMT_HOME=<directory where MGIZA++ is installed>
$CHASKI_HOME=<directory where Chaski is installed>
```

3.1. Pipeline 1: perform full training

The input of the pipeline is the source and target corpus file. Optionally the user can specify the word cluster files and ignore the word clustering step. In addition to the corpus files, the user also need to specify a root directory on HDFS where the user has full priviliges, and we denote is as `$ROOT`.

Chaski uses commandline interface and a configure file to get parameters, and supporting scripts are provided to make configuration and training easy. To train the model user need to follow the following three steps: 1) generate the config file, 2) modify the config file if necessary, 3), run the training script. To generate the config file for full training, just run:

```
$CHASKI_HOME/scripts/setup-chaski-full SOURCE-CORPUS \
                    TARGET-CORPUS $ROOT > chaski.config
```

and a config file `chaski.config` will be generated in current directory and then the user can fine-tune the parameters. After the parameter file is ready, the user can call the training script to start training:

```
$CHASKI_HOME/scripts/train-full chaski.config [first-step] [last-step]
```

There are two optional options `first-step` and `last-step` which can be used to resume training or bypass certain steps. The final output will be stored on HDFS:

```
$ROOT/moses-phrase      : Phrase table in Moses format
$ROOT/moses-reorder     : lexicalized reordering in Moses format
```

```
$ROOT/extract            : Extracted phrases
$ROOT/lexicon            : The lexicon tables in Moses format
$ROOT/training/S2T/Align : GIZA alignment directory, source-to-target
              /T2S/Align : GIZA alignment directory, target-to-source
```

3.2. Pipeline 2: phrase extraction only

If the user only wants to run phrase extraction, then in addition to source and target corpus files, the symmetrized word alignments must be supplied. Similar to full training pipeline, another script is used to set up config file:

```
$CHASKI_HOME/scripts/setup-chaski SOURCE-CORPUS
              TARGET-CORPUS ALIGNMENT $ROOT > chaski.config
```

and the script to run the training is:

```
$CHASKI_HOME/scripts/extract chaski.config [first-step] [last-step]
```

The output will be in the same directory as listed above, but it will not contain GIZA alignment directories.

3.3. Configuration

Limited by the length of the paper, we only list several important parameters the user should be aware of:

- `heap` The Java heap size for every job, the Hadoop installation may have limitations on the value, for large corpus you need to increase the value but it should not exceed the limitation imposed by the system.
- `memorylimit` The memory limitation for lexical translation table in the word alignment step, which is used to determine the size of chunks. Similarly the limitation should not exceed the limitation of Hadoop installation, but setting it too small will generate too many chunks and the overhead of loading parameters may impact the training speed.
- `train` Training sequence of distributed word alignment. The format of the training sequence is as follows: the number of iterations run on individual child is specified by characters `1`, `2`, `3`, `4` and `H`, and the global normalization is specified by `*`. For example `train=1*1*1*1*1*H*H*H*H*H*3*3*3*4*4*4*` will perform five model 1 iterations, five HMM iterations, and three model 3/4 iterations, and the normalization will take place after each iteration.

4. Experiments

4.1. Run time comparison

We compared running word alignment using MGIZA++ on quad-core Xeon CPU with running distributed word alignment using Chaski. The corpus used in the experiment is the GALE Arabic-English training corpus, which contains 6 million sentence

Table 1. Run time comparison of MGIZA++ and Chaski

System		Run time								
		Model 1		HMM		Model 3		Model 4		1-To-4
		Total	Iter	Total	Iter	Total	Iter	Total	Iter	
MGIZA	EN-AR	4.25h	0.85h	17.5h	3.50h	5.75h	1.15h	19.2h	3.85h	46.7h
	AR-EN	4.03h	0.80h	15.8h	3.15h	5.86h	1.17h	21.8h	4.35h	47.0h
Chaski	EN-AR	2.28h	0.46h	2.10h	0.42h	0.97h	0.32h	1.13h	0.38h	6.49h
	AR-EN	2.45h	0.49h	2.40h	0.48h	1.22h	0.41h	1.27h	0.42h	7.34h

pairs and 200 million words. We ran 5 model 1 iterations, 5 HMM iterations, 3 model 3 iterations and 3 model 4 iterations. We ran Chaski on Yahoo!'s M45 cluster, which has 400 nodes, each has 6 cores. The corpus is split into 125 chunks. Table 1 shows the run time comparison of MGIZA++ and Chaski. As we can see, we can cut the run time to less than 8 hours by using Chaksi.

We performed phrase extraction with Chaski and Moses on the same corpus, for Moses we used 16G memory in sorting, which is still not enough for loading all phrase pairs so external sort was triggered. The entire phrase extraction task took 21 hours, while with Chaski we finished the process in 43 minutes with 100 mappers and 50 reducers.

4.2. Translation result comparison

To compare the translation results, we used NIST MT06 evaluation set (1797 sentences about 50000 tokens) as tuning set and MT08 evaluation set (1360 sentences and about 45000 tokens) as test set, table 2 shows the BLEU scores of tuning and decoding using alignments and phrase table generated from different tools. "Phrase Table (Tune)" column lists the phrase table used in MERT and "Phrase Table (Test)" is the phrase table used in decoding. In the experiment a small tri-gram language model is used because we are mainly focus on the validity of the result rather than high BLEU score. As we can see, using phrase tables from Moses or Chaski has minimal difference due to different precision or float number formats, direct comparison on phrase table showed no phrase pair has different feature value if rounded to first four digits. Also, distributed word alignment outputs similar BLEU scores, although out of 6 million sentence pairs, 12.9 thousand sentence pairs have at least one different alignment link, the performance is generally unchanged.

5. Conclusion

In the paper we present a distributed training system, Chaski, for phrase based machine translation system runs on top of the Hadoop framework. The training time of word alignment is reduced from 47 hours to 8 hours and the time of phrase extraction/scoring from 21 hours to 43 minutes by using the system. The output phrase

Table 2. Translation Results

Word Aligner	Phrase Table (Tune)	Phrase Table (Test)	BLEU MT06	BLEU MT08
MGIZA	Moses	Moses	45.48	42.51
MGIZA	Moses	Chaski	45.40	42.51
MGIZA	Chaski	Chaski	45.75	42.46
MGIZA	Chaski	Moses	45.73	42.43
Chaski	Chaski	Chaski	45.33	42.49

tables are compatible with the Moses decoder. The system enables utilizing large clusters to train phrase-based machine translation models efficiently.

Acknowledgement

This work is supported by NSF Cluster Exploratory project (CluE- INCA, NSF08560), and we thank Yahoo! for providing M45 cluster for the research.

Bibliography

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. In *Computational Linguistics*, volume 19(2), pages 263–331, 1993.

Dean, Jeffrey and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

Dyer, Chris, Aaron Cordova, Alex Mont, and Jimmy Lin. Fast, easy, and cheap: Construction of statistical machine translation models with MapReduce. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 199–207, June 2008.

Gao, Qin and Stephan Vogel. Parallel implementations of word alignment tool. In *Proceedings of the ACL'08 Software Engineering, Testing, and Quality Assurance Workshop*, 2008.

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. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL'07*, pages 177–180, June 2007.

Och, Franz J. and Hermann Ney. A systematic comparison of various statistical alignment models. In *Computational Linguistics*, volume 1:29, pages 19–51, 2003.

Uszkoreit, Jakob and Thorsten Brants. Distributed word clustering for large scale class-based language modeling in machine translation. In *Proceedings of ACL-08: HLT*, pages 755–762, June 2008.

Venugopal, Ashish and Andreas Zollmann. 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, 2009.

Vogel, Stephan., Hermann Ney, and Christoph Tillmann. HMM based word alignment in statistical machine translation. In *Proceedings of COLING'96)*, pages 836–841, 1996.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 47–56

Tradubi: Open-Source Social Translation for the Apertium Machine Translation Platform

Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz

Dept. Llenguatges i Sistemes Informàtics, Universitat d'Alacant, Spain

Abstract

Massive online collaboration could become a winning strategy to tear down the language barriers on the web, and in order for this to happen appropriate computer tools, like reliable machine translation systems and friendly postediting interfaces, should be widely available. However, community collaboration should not only involve the postediting of machine translations, but also the creation of the linguistic resources needed to improve the translation engines. In this paper we introduce Tradubi, a free/open-source web application for social translation, whose aim is, firstly, to build a platform for collaboratively customising and improving rule-based machine translation systems and, secondly, to offer an environment for the postediting and subsequent sharing of raw machine translations. Currently, Tradubi is built upon the free/open-source Apertium machine translation engine. The application can be accessed at `tradubi.com` or downloaded and installed on a different server.

1. Introduction

The role of internet users has quickly evolved since the irruption of the web in the middle of the nineties: early passive consumers have become active *prosumers* (a word coined to refer to users which are both producers and consumers) of information. Under this view, internet companies *simply* build the spaces for interaction and users *colonise* them. This active role of users constitutes one of the main characteristics of what has been tagged as the *web 2.0* (O'Reilly, 2005).

However, in spite of the vast amount of contents uploaded to the *cloud* (another neologism which is commonly used as a synonym for internet) during the last years, linguistic barriers are still a significant obstacle to universal collaboration since they end up creating *islands* of content, only meaningful to speakers of a particular language.

Corresponding author: japerez@dlsi.ua.es
Cite as: Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz. Tradubi: Open-Source Social Translation for the Apertium Machine Translation Platform. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 47–56. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0012-8.

Massive online collaboration (involving not only professional translators but also amateurs) is probably the only force capable of tearing down these barriers (Garcia, 2009). The resulting scenario, which can be defined as *social translation*, will need efficient computer translation tools, such as machine translation (MT) systems or shared translation memories.

In the particular case of MT, collaboration should not only concern the postediting of raw translations, but also the creation of the linguistic resources needed by MT systems and the improvement of the translation engines. In this paper, we introduce for the first time Tradubi[1], a free/open-source web application whose aim is to ease these steps and become a platform for social translation. At the moment, Tradubi is built upon the Apertium free/open-source platform for rule-based MT (Forcada et al., 2009). With the help of Tradubi, users can create customised dictionaries for Apertium which focus on specific linguistic domains or which correct translation errors made by the default system. Tradubi allows every user or group of users to configure their own Apertium-based machine translator by defining a hierarchy of dictionaries to be used when translating texts.

Besides that, the last version of Tradubi includes a simple mechanism for the storage and management of the postedited translations. This feature is expected to improve in future versions so that users can work collaboratively on the postediting, refinement and publishing of translations.

Section 2 reviews some of the current approaches to social collaboration in the field of translation. Then, section 3 enumerates the main features of the current version of Tradubi. After that, some technical issues related to the development of Tradubi are discussed in section 4. The paper finishes with some conclusions and an overview of the features to be incorporated into future versions of the application.

2. Social Translation on the Web 2.0

There are a lot of web-based services for human translation. A selection of some of the the most relevant follows:

- Cucumis[2] is an online collaborative translation service based on an exchange policy: users gain points when they translate a document and these points are needed if they want to submit a text to be translated by other users (Cucumis' motto is "do you want to translate or to be translated?").
- Traduwiki[3] or Worldwide Lexicon[4] are similar to Cucumis but with a more open policy regarding who can translate or ask for a translation.

[1] Tradubi can be accessed at `http://tradubi.com`, and its source code can be downloaded from `http://tradubi.sourceforge.net`.

[2] `http://www.cucumis.org`

[3] `http://traduwiki.org`

[4] `http://www.worldwidelexicon.org`

- OneHourTranslation[5] is more business-oriented: users pay for translations and the company deducts a small commission from every transaction.

None of the previous sites enforce any particular tool to carry on the translations. A different group of web applications like Wiktionary[6] or Lingro[7] are focused on the collaborative building of dictionaries and terminological databases.

Web-based *tools* for translation can also be found. For example, the recently launched Google Translator Toolkit[8] allows users to create, maintain, use and share translation memories and terminological databases, as well as combining them with statistical MT through a specialised interface for translation inspired on the one popularised by traditional translation memory management systems; users have access to the statistical MT system but they cannot modify directly its behaviour. In connection with the system presented in this paper, that is, one dealing with the configuration of rule-based MT system, some similar approaches can be found in the literature, as, for example, the translation environment *Yakushite Net* (Murata et al., 2003). Our proposal is the first free/open-source and the first to focus on the expanding Apertium platform.

3. Current Features of Tradubi

Tradubi is an Ajax-based (Garret, 2005) web application, that is, an application which can be run in a browser without requiring installation of any additional plugin. The client side (mostly, the interface) of the application is therefore encoded in JavaScript (see 4.2 for more details) and communicates with a server responsible for the tasks which cannot be executed in the browser. This follows an emerging trend on the web where applications are moving from the desktop to the *cloud*.

Users of Tradubi create customised dictionaries of translation units, which consist of a word or sequence of words in the source language and their corresponding translation in the target language (for example, the English–Spanish translation unit *glucose-6-phosphate isomerase/glucosa-6-fosfato isomerasa*). Note that in the current version no morphological information is attached to the words, which makes it easier for non-experts to add new entries but might require to add all the lexical variations of a word in some cases. User dictionaries and translation units can be used in the following ways:

- *Creation:* users can add new translation units to the engine.
- *Maintenance:* the translation units can be modified or deleted at any time.
- *Hierarchy definition:* a group of user dictionaries (for example, for different linguistic domains) can be used in new translations; in order to avoid conflicts, users can define a hierarchy of these dictionaries.

[5] http://www.onehourtranslation.com

[6] http://www.wiktionary.org

[7] http://lingro.com

[8] http://translate.google.com/toolkit/

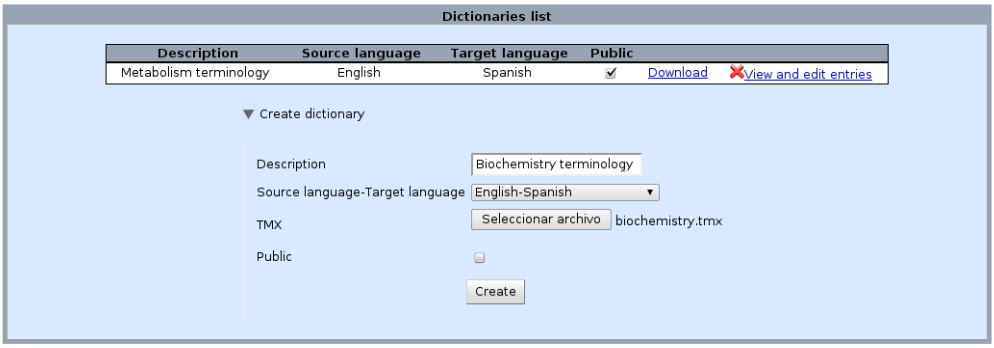

Figure 1. A screenshot of Tradubi showing the creation of a new English-Spanish user dictionary intended for biochemical terms. The dictionary will be initially fed with translation units from the biochemistry.tmx *file. Data will be non-public (private or shareable). The list of current available dictionaries is shown on the top (in this case, a public dictionary with terminology about metabolism).*

- *Sharing:* a user dictionary can be tagged as public, private or shared (in read-only mode at this moment) with other users; when defining a dictionary hierarchy, every available dictionary can be considered.
- *Recommendation:* Tradubi can suggest before translation a dictionary or a set of dictionaries for a particular source text according to the number of words in the text found in the dictionaries.
- *Import/export:* the translation units in a dictionary can be imported or exported using the Translation Memory eXchange[9] (TMX) standard format.
- *Collaborative creation:* shared dictionaries may received translation units from every user with permissions; this feature will allow for the collaborative creation of dictionaries, but it is not implemented in the stable version of Tradubi yet.

Apart from this, postedited translations can be stored and retrieved at any time; this feature will evolve to a system for collaborative postediting and dissemination of translations.

Figures 1 to 3 show some screenshots of the application with some additional comments.

[9] `http://www.lisa.org/standards/tmx/`

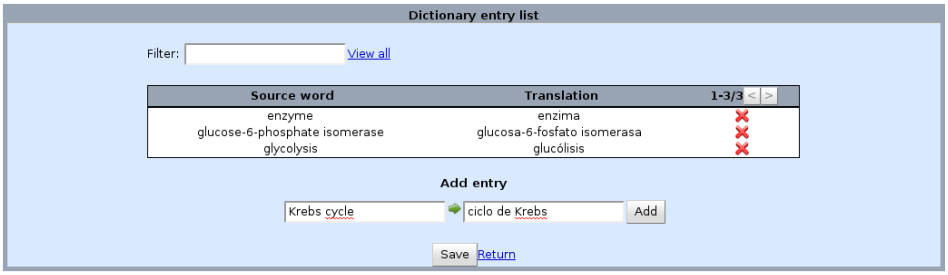

*Figure 2. A screenshot of Tradubi showing the addition of a new translation unit (*Krebs cycle/ciclo de Krebs*). The dictionary already contains three translation units which can be modified or deleted. The* save *button triggers the compilation of the dictionary to the binary form used by Apertium.*

4. Technical Issues

Tradubi's design, development and deployment requires dealing with a number of technical issues which are discussed in this section.

4.1. License Choice

Tradubi is not only available through a public web server, but also as a free/open-source program which can be downloaded, installed and modified by everyone. It is licensed under version 3 of the GNU Affero General Public License[10] (AGPL). This license is fully compatible with the GNU General Public License (GPL) and equally proposed by the Free Software Foundation, which in fact recommends[11] that "developers consider using the GNU AGPL for any software which will commonly be run over a network". AGPL has been suggested as a means to close a *loophole* in the ordinary GPL which does not force organisations to distribute derivative code when it is only deployed as a web service.

Choosing AGPL is a little big controversial (O'Grady, 2009) since this license adds a new constraint to the well-established GPL license. We consider, however, that in the web 2.0 era and with the traditional model of software distribution gradually losing ground to the cloud computing model, AGPL should be being adopted by a higher number of free/open-source projects.

[10] `http://www.gnu.org/licenses/agpl-3.0.html`

[11] `http://www.fsf.org/licensing/licenses/`

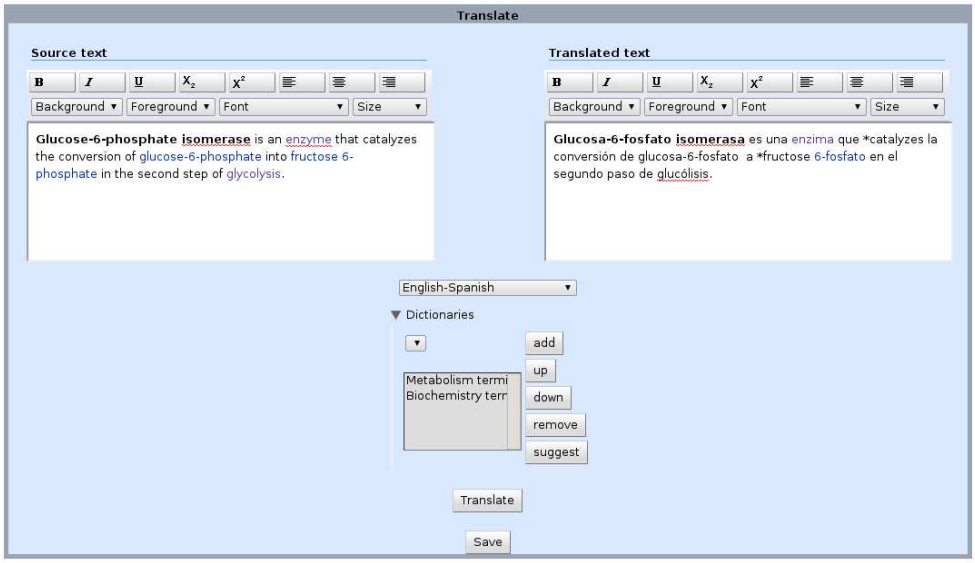

Figure 3. A screenshot of Tradubi showing a translation with two user dictionaries (one on metabolism, with higher precedence, and a second one on biochemistry). The two dictionaries include a different translation for the word glycolysis *(*glucólisis *and* glicolisis*) but the one in the first dictionary (*glucólisis*) has been chosen because of the hierarchy defined in this case. Apart from this explicit choice of dictionaries, the* suggest *button automatically selects the most appropriate dictionary according to the source text. The resulting machine translated text on the right is ready to be postedited and then saved.*

4.2. Programming Language and Framework

Tradubi client is mostly written in Java with the help of the Google Web Toolkit[12] (GWT). GWT is a free/open-source framework for developing web applications. At the core of the framework is a compiler which translates the code written for the client in Java to JavaScript code which runs flawlessly in current browsers. GWT simplifies the development and debugging of Ajax-based web applications which require asynchronous remote procedure calls, history management, bookmarking, internationalisation or code splitting.

[12] `http://code.google.com/webtoolkit/`

The code for the server side is written in Java as well. The project code consists (as of current version) of 135 classes and around 13 000 lines of code.

4.3. Data Portability and Accessibility

According to the DataPortability Project,[13] *data portability* is the "ability for people to reuse their data across interoperable applications". This is key feature which the web 2.0 should embrace in order to mitigate the undesirable consequences of *walled gardens*. Import and export of the dictionaries in TMX format allow Tradubi users to seamlessly move their data, for example, to Google Translator Toolkit.[14]

In addition to this, Tradubi users may log into the application using an existing OpenID account.[15] User interface and dynamic content are more accessible thanks to the adoption of the WAI-ARIA[16] standard.

4.4. Apertium Server

Communication between Tradubi and the Apertium engine is done via an already implemented scalable architecture (Sánchez-Cartagena and Pérez-Ortiz, 2009) for Apertium. This architecture consists of a *router server* which forwards incoming translation requests to one or more slaves running Apertium instances. Our web application sends requests to the router through the Remote Method Invocation (RMI) protocol; although a convenient Application Programming Interface (API) is also available, we chose to use RMI directly since Tradubi and the scalable translation system are both written in Java.

4.5. Integration with Apertium

As already commented, Tradubi allows every user or group of users to configure their own Apertium-based machine translator by defining a hierarchy of dictionaries to be used when translating texts: matched entries in a dictionary at level i of the hierarchy take precedence over any other match found in a dictionary at level j with $i < j$, default system dictionaries being at the highest level (that is, they have the minimum precedence).

User dictionaries are specified, compiled and accessed as regular Apertium monolingual dictionaries (Forcada et al., 2009), except for the fact that no morphological information is attached to the entries. Originally, every translation unit is coded in

[13] `http://www.dataportability.org`

[14] Note that, at the time of writing, Google Translator Toolkit does not allow users to export their own data.

[15] `http://openid.net`

[16] `http://www.w3.org/WAI/intro/aria`

XML inside an *e* element containing the source and target words; the following is an excerpt of an English–Spanish user dictionary:

```
<e><p>
  <l>glucose-6-phosphate<b/>isomerase</l>
  <r>glucosa-6-fosfato isomerasa</r>
</p></e>
```

The XML file is then compiled to a binary form by means of the *lttoolbox* library included in Apertium. The binary version of the dictionaries implement a finite-state transducer (Roche and Schabes, 1997) which is used to efficiently detect and translate the source words. This compilation is done as soon as the user clicks on the *save* button after introducing or modifying a set of translation units (see figure 2); therefore, the new units are ready immediately for new translations.

The resulting transducer is inserted in the Apertium pipeline between the part-of-speech tagger and the structural transfer module. This way, the tagger has more information for disambiguation since it can consider the lexical categories of words in the default dictionaries of Apertium which, however, are going to be translated with a user dictionary. If a user defines a hierarchy of dictionaries the system is set up as a cascade of modules that successively search for the words in the source text and keep the first translation found.

Some problems arise when a translation unit contains a word that is part of a *multiword* in the default system dictionaries. For example, if default dictionaries contain an entry for *an arm and a leg* and a user dictionary contains the translation unit *leg/etapa*, then the word *etapa* will never appear in the target text when translating a source sentence which includes the multiword.

4.6. Compilation and Installation

Source code can be downloaded from the SVN repository of Tradubi located at *Sourceforge.net*.[17] It includes documentation with additional instructions on how to compile and install the application.

5. Future Work on Tradubi

Tradubi is still in its early stages of development, but with some of the following improvements we expect it to become a mature and stable framework for social translation.

It is worth studying alternative places of insertion into the Apertium pipeline of the new modules dealing with user dictionaries. Currently they are located just before the

[17] `http://tradubi.sourceforge.net`

structural transfer module, but locating them in other positions (for example, before the morphological analyser) could result in the overcoming of the multiword problem (see 4.5) while keeping functionality the same.

We also plan to consider the inclusion of other MT engines in addition to Apertium. The open-source nature of Apertium has allowed us to easily insert the new modules for user dictionaries in the middle of its pipeline, but it might be very interesting to research how to extend these modules to engines with no source code available and which can be accessed online through web services only. This would require to consider how to isolate the words found in the user dictionaries from the rest of the text which should be translated by the MT engine.

In harmony with the idea of adopting the principles of open data (see 4.3), we will also implement an option for downloading for local installation a package with the Apertium engine and all the linguistic data and user dictionaries making up a particular translator that a user has configured online.

The current simple interface for postediting will evolve into a more friendly interface which benefits from information extracted from the MT engine in a way similar to recent proposals in the statistical MT field (Koehn, 2009). The information collected from the interaction of the users with the postediting interface will also be used to improve the linguistic data (both dictionaries and structural transfer rules) of the Apertium-based translators in a similar manner to the *Translation Correction Tool* (Font-Llitjós et al., 2005).

Finally, more social features could be added to the application.

6. Conclusions

This is the first paper to introduce Tradubi, a free/open-source Ajax-based web application for the collaborative configuration of rule-based MT systems. Currently, Tradubi works as a layer over Apertium, allowing users to create and use hierarchies of dictionaries which override default system dictionaries in case of conflict. We expect to augment the functionalities of Tradubi so that it becomes a powerful application for social translation.

7. Acknowledgements

This work has been partially funded by Spanish Ministerio de Ciencia e Innovación through project TIN2009-14009-C02-01.

Bibliography

Font-Llitjós, Ariadna, Jaime Carbonell, and Alon Lavie. A framework for interactive and automatic refinement of transfer-based machine translation. In *Proceedings of EAMT 10th Annual Conference*, 2005.

Forcada, Mikel L., Francis M. Tyers, and Gema Ramírez-Sánchez. The Apertium machine translation platform: five years on. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 3–10, 2009.

Garcia, Ignacio. Beyond translation memory: Computers and the professional. *The Journal of Specialised Translation*, 12:199–214, 2009.

Garret, Jesse James. Ajax: A new approach to web applications. *AdaptivePath.com,* `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, 2005.

Koehn, Philipp. A Web-Based interactive computer aided translation tool. In *Proceedings of the ACL-IJCNLP 2009, Software Demonstrations*, pages 17–20, 2009.

Murata, T., M. Kitamura, T. Fukui, and T. Sukehiro. Implementation of collaborative translation environment: YakushiteNet. In *Proceedings of MT Summit IX*, 2003.

O'Grady, Stephen. AGPL: Open source licensing in a networked age. *RedMonk.com,* `http://redmonk.com/sograady/2009/04/15/open-source-licensing-in-a-networked-age/`, 2009.

O'Reilly, Tim. What is web 2.0. *O'Reilly Network,* `http://oreilly.com/web2/archive/what-is-web-20.html`, 2005.

Roche, Emmanuel and Yves Schabes. *Finite-state language processing*. MIT Press, 1997.

Sánchez-Cartagena, Víctor M. and Juan Antonio Pérez-Ortiz. An open-source highly scalable web service architecture for the Apertium machine translation engine. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 51–58, 2009.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 57–66

Adding Multi-Threaded Decoding to Moses

Barry Haddow

School of Informatics, University of Edinburgh, Informatics Forum, 10 Crichton Street, Edinburgh, Scotland, EH8 9AB

Abstract

The phrase-based translation system Moses has been extended to take advantage of multi-core systems by using multi-threaded decoding. This paper describes how these extensions were implemented and how they can be used, as well as offering some experimental measurements of the potential speed-ups available. Details are also provided of how the multi-threaded Moses library is used to create the Moses server, a platform for building online translation systems.

1. Introduction

A recent trend in computing has been the growth in popularity of *multi-core* processors, able to execute several processes simultaneously. Ordinary desktop and laptop machines are frequently equipped with dual-core processors while servers may have one or more 8-core processors. In order to take advantage of this parallel computing capability, software can be developed to execute with multiple *threads*. Whilst both threads and processes are units of execution, the difference between the two is that threads share the same address space, meaning that multiple threads can access the same in-memory data structures. The consequence is that threads can cooperate more tightly to accomplish a task, but also that the developer must take more care to ensure that common data structures are not damaged by interleaved instructions.

The aim of this paper is to describe some recent modifications to the Moses[1] decoder (Koehn et al., 2007) which enable it to take advantage of this parallel computing capability by decoding several sentences simultaneously in separate threads. Within the typical machine translation (MT) research environment, the main advantage of a

[1] Available under the LGPL from `http://sourceforge.net/projects/mosesdecoder/`

Corresponding author: bhaddow@inf.ed.ac.uk
Cite as: Barry Haddow. Adding Multi-Threaded Decoding to Moses. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 57–66. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0006-6.

multi-threaded decoder is that it can make more efficient use of the available hardware, enabling quicker decoding. Since the most widespread method for optimising statistical machine translation systems, minimum error rate training (mert) (Och, 2003), involves decoding the tuning set multiple times, improvements in decoding speed lead to faster experimental turnarounds.

The traditional method of parallelising decoding (as implemented in Moses by the `moses-parallel.pl` script) was to split the input file into equal sized segments and send each segment to a separate process, probably running on a separate machine. This method requires a cluster of machines running some kind of job scheduling software (such as Sun grid engine), requiring specialist knowledge to install and administer. It also requires each machine to have access to the translation, language and reordering models, and to have sufficient RAM for the decoder to be able to load them into memory. With the increasing size of the models that are used in MT research, copying these across a network and providing sufficient RAM are non-trivial tasks. The advantage of using threads for parallel decoding is that, since all the parallel execution takes place in the same process, only one copy of each of the models needs to be loaded into memory. Furthermore, it is easier to balance the decoding load between threads than between different processes, as they can cooperate more closely.

Parallel decoding is also essential for the provision of on-line translation services. In this setting, it is clearly undesirable for one user to be blocked whilst another user's translation job is running, and for translating larger blocks of text (such as web pages) it would be useful if some of the sentences could be translated in parallel. Adding multi-threading to the Moses library meant that the decoder could be embedded within a server which is able to process multiple simultaneous requests. Of course, to create a truly scalable online translation system, it is also necessary to allow translation to be spread across multiple machines (Sánchez-Cartagena and Pérez-Ortiz, 2009), as adding more machines an easier way of scaling hardware if the current server's capacity has been reached. Nevertheless, a multi-threaded moses server is an important component in a moses-based online translation system, since it can take advantage of multi-core servers.

The main disadvantage of multi-threaded software is that it can be more complicated to develop, and leads to a new types of bugs which may be difficult to diagnose. In this paper, the techniques used to add thread safety to an existing decoder (namely, Moses) will be discussed, with the aim of providing guidance to others working on similar engineering problems.

The paper is organised as follows: in the following section, techniques for safe multi-threaded programming are described, while Sections 3 and 4 explain the design of multi-threaded Moses and the Moses server, respectively. In Section 6 some experimental results are presented showing the speed-ups possible when decoding with multi-threaded Moses, whilst Section 7 offers some conclusions and suggestions for future developments.

2. Techniques for Multi-threaded Programming

The aim of this section is to briefly introduce some of the concepts and techniques used in multi-threaded programming. It is not meant to be a comprehensive treatment of the topic, merely to provide sufficient background for the design description in the following section.

In most operating systems, programs are executed as *processes* which are separate units of execution (as seen by the scheduler) and have separate address spaces, so they cannot normally access each other's data. A process may, however, have one or more separate *threads*, which are also units of execution with their own call stack, but share the same address space. On a single-processor, single-core machine, threads are mainly used so that the process can continue doing work whilst it is waiting on another task (typically input/output) to complete. However on today's multi-core and multi-processor machines, genuine parallelism is possible.

Allowing multiple threads of execution to access the same memory space is potentially dangerous and can easily lead to memory corruption. To allow safe concurrent access to data structures, a device called a *mutex* (mutual exclusion), or a *lock*, is used to protect critical sections of code so that only one thread is allowed to execute it at any one time. One particularly useful type of mutex is a *reader-writer lock*, which has two modes of locking; one for reading which allows many threads to access the critical section simultaneously, and one for writing where only one thread is allowed to access it.

If mutexes are not used correctly then performance can suffer due to either lock contention or deadlock. The former is where threads hold on to mutexes for longer than is necessary, thus reducing performance because many other threads may be waiting on the mutex to continue performing their tasks. The latter situation can arise, for example, where a first thread acquires lock `A` and then lock `B`, whilst a second thread attempts to acquire the same locks in the opposite order. Depending on the timing of lock acquisition, each thread can be left waiting for the lock that the other one holds, and so neither will be able to continue executing.

As an alternative to the coordinated sharing of data using mutexes, it is sometimes appropriate for each thread to have its own data, for example to provide a thread-specific cache. On many platforms developers implement this using a construct called *thread local storage*. Conceptually, this can be thought of as a common pointer, which when dereferenced returns a block of memory which is unique to the calling thread.

It is possible to create and destroy threads whenever needed, however, creating and destroying threads can be expensive (if they use a lot of thread specific data) and it may be necessary to limit the number of threads active at any time. To avoid constant creation and destruction, threads are often organised into a *thread pool*, whose size may be fixed or subject to upper and lower bounds. A work queue can be used in conjunction with a thread pool to allocate work to the threads, by queueing up the tasks and allocating them to the next available thread.

Programming languages and operating systems differ in the amount of support they offer for multi-threaded programming. In general, Java has very good multi-threaded support, having been created as a multi-thread enabled platform right from the start, and possessing a broad range of thread synchronisation primitives, as well as thread-safe data structures and classes for implementing typical threaded programming patterns. In C++ the multi-threaded support is not as good, with no single standard library for multi-threaded programming, and many platform specific thread libraries. However, there are some mature cross-platform libraries for C++ which include all the appropriate multi-threaded programming primitives, such as ACE[2] (Adaptive Communications Environment) and boost[3]. In addition, there is OpenMP[4], a cross-platform API supported by many leading software vendors which is implemented mainly using compiler directives. In multi-threaded Moses, the boost libraries were used since they offer the required primitives in a cross-platform library which is steadily being incorporated into the C++ standard.

3. Multi-threaded Moses

The aim of this section is to explain the changes that were made to Moses in order for it to support multi-threaded decoding. The threading model adopted for multi-threaded Moses assigns each sentence to a distinct thread so that each thread works on its own decoding task, but shares models with the other threads. This design was chosen to minimise the data sharing between threads.

In making the required changes for multi-threading, one of the considerations was to cause as little disruption to the existing codebase as possible, so the design decisions are not necessarily the same as those that would be employed when building a new piece of software. It was important not to introduce extra dependencies to the Moses build, except where necessary, so the thread-safe version of the Moses library is only built when the appropriate compiler directives are switched on. The work involved in adding multi-threaded decoding to Moses can be divided into two parts; updating the Moses libraries to be thread-safe, and adding the thread creation and management to the Moses mainline.

3.1. Moses Library

To enable the multi-threaded decoding, the Moses libraries need to ensure that, when two different threads are processing their respective sentences, they do not attempt to modify data structures potentially being used by the other threads. The principal shared data structures used in decoding are language models, translation

[2] http://www.cse.wustl.edu/~schmidt/ACE.html

[3] http://www.boost.org

[4] http://openmp.org/wp/

models and reordering models, and at first sight one might think that all the decoder needs to do is read from these data structures, in which case there would be no issue with simultaneous access. However the extensive use of caching within Moses, necessary to reduce levels of disk access during decoding, meant that the data structures representing the models were not necessarily read only. Furthermore, Moses tended to rely on the global singleton object `StaticData` to store data connected with the translation process, even if it was only relevant for one sentence.

The first strategy employed to ensure the thread-safety of the Moses libraries was to use the `Manager` object to store sentence specific data, rather than `StaticData`. An instance of the `Manager` is created for each sentence to be translated, and only contains data relevant to that particular sentence. So in the 'thread per sentence' model employed in multi-threaded Moses, these objects can only be accessed by one thread at a time. The disadvantage of using the `Manager` object to store sentence-specific data is that it must be made available at all points at which this data is needed, thus cluttering up interfaces. In the thread-safe Moses, the `Manager` is now responsible for the pre-loaded portion of the translation table pertaining to its sentence, as well as certain debug data such as timing information.

The translation table (phrase dictionary) in Moses can either be loaded it to memory or utilised in a 'binarised' (on-disk) mode. The former presents no thread-safety issues since it is just loaded into memory at decoder start-up, and is used in a read-only fashion. However, with large translation models it is usual to compile them into a binary format and use them in the on-disk mode, which means that some caching is necessary to reduce the amount of disk access. The system-wide disk cache would be of some help here, but a cache that works at the phrase level is more effective.

The binarised translation model is controlled by the `PhraseDictionaryTree` class which is really just a wrapper for the `PDTImpl` class, the actual implementation. Since the latter is is a read-write data structure, it needed modification to allow concurrent access, and in order to minimise the code changes involved it was decided to use thread specific data to make sure that there was only ever one `PDTImpl` object per thread. The thread specific data class in the boost library has the advantage that it has the same interface as an `auto_ptr`, making it easy to switch between two using compiler directives.

A third solution to the problem of allowing multiple threads to simultaneously access data structures was to use mutexes. For example, the `FactorCollection` object (essentially a vocabulary cache) is now protected by a reader-writer lock so that multiple threads can read from it at any one time, but if a thread wishes to write to it then it must obtain an exclusive lock. For the translation options cache held in the global `StaticData` object, a single mutex is used to synchronise access to the cache. As this is an LRU (least recently used) cache, it must update a timestamp every time an item in the cache is accessed, so a reader-writer lock is not appropriate here.

3.2. Mainline

In order to run multi-threaded decoding, the Moses mainline must create threads and organise the assignment of decoding work to threads. Due to the multiplicity of input/output options in the existing mainline, it was decided that it would be easier to create a new multi-threaded mainline (`MosesMT.cpp`) rather than updating the old one. This has resulted in some undesirable duplication of code, complicating regression testing, which hopefully will be resolved in a future refactoring.

A UML sequence diagram for the important parts of the new Moses mainline is shown in Figure 1. The mainline creates a `ThreadPool` object whose job it is to manage a pool of worker threads. The specified number of threads is created on construction and then jobs are submitted to the pool using the `Submit()` method until the `Stop()` method is called which causes the pool to stop accepting new work, flush the queue and stop all the threads. The unit of work processed by the thread pool is represented by a `Task` object, which in the multi-threaded decoder contains a single sentence to be translated. The tasks are queued up in the pool and as threads become available, they pop a task off the queue and execute it.

When multi-threaded Moses is processing a file, the file is read in, split into lines, and placed in the `ThreadPool`'s queue as a series of `Task` objects. Since these may be executed out of order, it is necessary to put the translated sentences into the appropriate order before outputting them. This reordering is performed by the `OutputCollector` class which uses the input line number to order the sentences correctly.

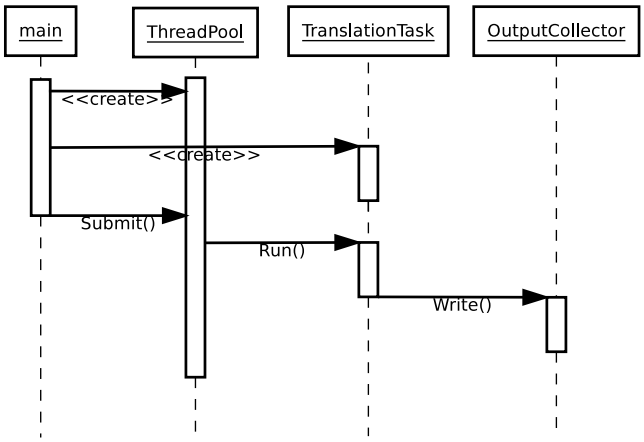

Figure 1. UML Sequence diagram for multi-threaded Moses mainline

4. Moses Server

The main purpose of the Moses server is to enable network access to a Moses-based translation system, for example to build an on-line demo. Making the server multi-threaded offers the advantage that it can process translation requests from more than one user simultaneously, and also it can decode multiple sentences in batches, for example when translating a web page.

The Moses server use the xmlrpc[5] protocol to communicate with its clients. This protocol has the advantage of having mature implementations available in many programming languages; the Moses server has been used with clients written in java, perl, python and php. The specific implementation used in Moses is xmlrpc-c[6].

Since the xmlrpc implementation takes care of managing the server infrastructure, for example listening for client requests and running a thread pool to deal with these requests, the Moses server code only has to implement the remote procedure calls (rpc). Currently the only call that the Moses server implements is the `translate()` call, which receives an input sentence in its `text` field, and returns the translated text in the same field. If the `align` flag is switched on in the method call then the phrase alignment is returned as a sequence of `(target-start, source-start, source-end)` index triples, in target order.

5. Usage

Using multi-threaded Moses is straightforward. The new mainline (`mosesmt`) is intended as a drop-in replacement for the existing mainline. It responds to exactly the same arguments as `moses` and adds a `-threads n` argument to specify the number of threads. Increasing the verbosity of multi-threaded Moses is not recommended as some of the debug code uses non-threadsafe global variables, and the debug messages will be interleaved and difficult to read anyway.

The Moses server mainline (`mosesserver`) also accepts all the usual Moses arguments and adds two of its own. The argument `--server-port n` is used to specify the port on which the server listens, and the `--server-log` can be used to specify a log file for the server to write to. For extra diagnostic information, set the `XMLRPC_TRACE_XML` environment variable before launching the server.

6. Experiments

In this section the results of some timing experiments are presented, comparing multi-threaded and single-threaded Moses. The experiments were run on a Dell PowerEdge server with 4 Intel Xeon Quad Core processors (so it has 16 cores), and 32GB RAM.

[5] `http://www.xmlrpc.com/`

[6] `http://xmlrpc-c.sourceforge.net/`

For the first set of experiments, decoding speed of single and multi-threaded Moses was directly compared, using a translation model similar to the Edinburgh French-English submission for the WMT2009 shared task (Callison-Burch et al., 2009; Koehn and Haddow, 2009). This includes translation models and reordering models trained on all the shared task parallel data, plus a language model trained on the English side of this data, interpolated with the monolingual news data. The translation and reordering models were used in binarised (on-disk) format, but the language model was loaded into memory.

The experiment consisted of decoding the news test set from this shared task (3027 sentences) using plain (single-threaded) Moses, and using multi-threaded Moses whilst varying the number of threads from 2 to 6. In order to account for the fixed costs of loading the models into memory and initialising other data structures, a decoding run was also done for one sentence. Decoding was repeated five times for each type of decoder. The mean times (in seconds) are shown in Table 1.

Decoder	Full corpus	One sentence	Difference	sd(Difference)
Plain Moses	4677	282	4395	623
Moses MT 2 threads	3292	283	3009	505
Moses MT 3 threads	2024	284	1740	154
Moses MT 4 threads	1781	283	1498	100
Moses MT 5 threads	1591	278	1313	37
Moses MT 6 threads	1492	278	1214	45

Table 1. Decoding time (in seconds) for single and multi-threaded decoders, averaged over five runs. The second last column is the difference between the first two, in other words the time to decode 3026 sentences not including start-up and shut-down times. The final column shows the standard deviation of this 3026 sentence time.

From Table 1 it can be seen that there is around a speed increase of around 3.5 going from single-threaded Moses to multi-threaded Moses with 6 threads. Whilst this speed-up is clearly useful, the question arises of why there isn't a six-fold increase in speed. The most likely answer to this question is some sort of resource contention; in other words the six threads are not spending all their time decoding but spending some time waiting for other threads to release a resource. One possible type of resource contention is lock contention, where threads spend time in a block state waiting for other threads to release locks, however the only locks used during the decoding are those on the translation options cache, and running experiments with this cache removed results in similar timing behaviour. It is also possible, depending on the hardware architecture, that there is resource contention at the RAM or disk level,

since decoding requires a substantial amount of data to be accessed from the different models employed.

The next timing experiment compares minimum error rate training (mert) runs using single-threaded and multi-threaded Moses. This experiment uses the French-English europarl corpus (Callison-Burch et al., 2009) for training the translation model and 5-gram language model, with the 2000 sentence dev2006 corpus for tuning, and the test2007 and test2008 corpora for testing. The tuning runs were done on the same machine as the first set of experiments, although because of the length of these experiments it was not possible to ensure that the machine remained unloaded throughout this time. Table 2 shows the timings for single-threaded Moses, and Table 3 shows the corresponding timings for multi-threaded Moses, demonstrating around a two-fold speed-up in mert when using 4 threads.

Run	Iterations	Time	Time per Iteration	Bleu
1	17	2054	120.8	33.4
2	12	1258	104.8	33.3
3	14	1362	97.3	33.3
4	14	1172	83.7	33.3
5	16	1283	80.2	33.3
mean	14.6	1425	97.4	33.3

Table 2. MERT times for single-threaded Moses, in minutes

Run	Iterations	Time	Time per Iteration	Bleu
1	15	735	49.0	33.3
2	23	1320	57.4	33.3
3	8	319	39.9	33.5
4	15	615	47.7	33.4
5	10	456	45.6	33.4
mean	14.2	689	46.6	33.4

Table 3. MERT times for multi-threaded Moses (4 threads), in minutes

7. Conclusions and Future Work

This article has described extensions to the Moses decoder which permit multi-threaded decoding, and also allow Moses to be used as a server to run an online

translation system. Experimental results demonstrated that using 6 threads can speed up decoding by 3.5 times, and a two-fold speed-up in mert was demonstrated, when using a multi-threaded decoder with 4 threads. Further investigation is required to determine why the speed of decoding is not linear in the number of threads. The other outstanding task in multi-threaded Moses is to make the generation tables (used in some factored models) thread-safe; these can be addressed using the same techniques as the translation tables.

The Moses server is already being used successfully in the University of Edinburgh's demo site[7]. A limitation of the current server is that a separate server is required for each language pair so, for instance, to deploy both French-English and German-English systems, each server must load its own copy of the English language model. A proposed update to the Moses server would be to allow *configuration switching*, where one server would be able to run more than one translation system, with the choice of translation system to translate a given sentence would be selected by an rpc argument. This arrangement would save on the RAM used to run multiple Moses servers on the same host with the same target language.

Bibliography

Callison-Burch, Chris, Philipp Koehn, Christoph Monz, and Josh Schroeder, editors. *Proceedings of the Workshop on Statistical Machine Translation*, 2009.

Koehn, Philipp and Barry Haddow. Edinburgh's submission to all tracks of the WMT 2009 shared task with reordering and speed improvements to Moses. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 160–164, 2009.

Koehn, P., H. Hoang, A. Birch Mayne, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demonstration Session*, pages 177–180, 2007.

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

Sánchez-Cartagena, Víctor M. and Juan Antonio Pérez-Ortiz. An open-source highly scalable web service architecture for the Apertium machine translation engine. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 51–58, 2009.

[7] `http://demo.statmt.org`

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 67–76

Free/Open-Source Resources in the Apertium Platform for Machine Translation Research and Development

Francis M. Tyers[a], Felipe Sánchez-Martínez[a], Sergio Ortiz-Rojas[b], Mikel L. Forcada[a c]

[a] Grup Transducens, Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, E-03071 Alacant, Spain
[b] Prompsit Language Engineering, Av. St. Francesc d'Assís, 74, 1r-L, E-03195 l'Altet, Spain
[c] CNGL, Dublin City University, Dublin 9, Ireland

Abstract

This paper describes the resources available in the Apertium platform, a free/open-source framework for creating rule-based machine translation systems. Resources within the platform take the form of finite-state morphologies for morphological analysis and generation, bilingual transfer lexica, probabilistic part-of-speech taggers and transfer rule files, all in standardised formats. These resources are described and some examples are given of their reuse and recycling in combination with other machine translation systems.

1. Introduction

Apertium (`http://www.apertium.org`) is a free/open-source (FOS) platform for creating rule-based machine translation systems (Forcada et al., 2009). There are currently stable data for 21 language pairs available within the platform. Resources within the platform take the form of finite-state morphologies for morphological analysis and generation, bilingual transfer lexica, probabilistic part-of-speech taggers and transfer rule files, all in standardised formats. These resources are described and some examples are given of their reuse and recycling in combination with other machine translation systems.

This article is organised as follows: section 2 describes the Apertium engine; section 3 describes the current status of the resources available in the platform; section 4

Corresponding author: fsanchez@dlsi.ua.es
Cite as: Francis M. Tyers, Felipe Sánchez-Martínez, Sergio Ortiz-Rojas, Mikel L. Forcada. Free/Open-Source Resources in the Apertium Platform for Machine Translation Research and Development. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 67–76. ISBN 978-80-904175-4-0.
doi: 10.2478/v10108-010-0015-5.

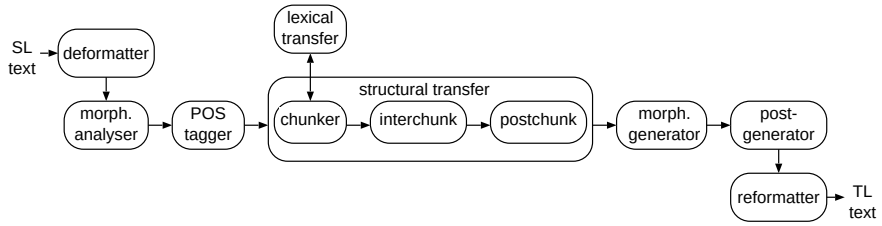

Figure 1: The modular architecture of the Apertium MT platform.

gives some details of ways these resources can be re-used within other machine translation systems, finally section 5 gives some directions of future work and discussion.

2. The Apertium platform

A very brief outline of Apertium will be given here. Turn to existing descriptions, such as Armentano-Oller et al. (2006) and Forcada et al. (2007), for details.

The Apertium platform provides: (a) A FOS modular shallow-transfer MT *engine* with text format management, finite-state lexical processing, statistical lexical disambiguation, and shallow structural transfer based on finite-state pattern matching; (b) FOS *linguistic data* in well-specified XML formats for a wide variety of language pairs; and (c) FOS tools such as *compilers* to turn linguistic data into a fast and compact form used by the engine and software to learn disambiguation or structural transfer rules, and (d) extensive documentation on usage.[1] The Apertium engine is a pipeline or assembly line consisting of the following stages or modules (see figure 1):

- A *deformatter* which encapsulates the format information in the input document as *superblanks* that will then be seen as blanks between words by the rest of the modules.
- A *morphological analyser* which segments the text in surface forms ("words") and delivers, for each surface form, one or more *lexical forms* consisting of *lemma*, *lexical category* and morphological inflection information. It reads a finite-state transducer (FST) generated from a source-language (SL) morphological dictionary (MD) in XML.
- An optional *constraint grammar*[2] (Karlsson et al., 1995) to reduce or remove entirely part-of-speech (PoS) ambiguity before the statistical PoS tagger, and to provide syntactic and semantic labelling.

[1] Documentation on a wide variety of development and usage scenarios can be found on the Apertium Wiki (`http://wiki.apertium.org/`).

[2] `http://beta.visl.sdu.dk/constraint_grammar.html`

- A *statistical PoS tagger* which chooses, using a first-order hidden Markov model (HMM: Cutting et al. (1992)), the most likely lexical form corresponding to an ambiguous surface form, as trained using a corpus and a tagger definition file in XML.
- A *lexical transfer* module which reads each SL lexical form and delivers the corresponding target-language (TL) lexical form by looking it up in a bilingual dictionary in XML using a FST generated from it.
- A *structural transfer*, generally consisting of three sub-modules (some language pairs use only the first module and some others call more than three, see below):
  - A *chunker* which, after invoking lexical transfer, performs local syntactic operations and segments the sequence of lexical units into chunks. A chunk is defined as a fixed-length sequence of lexical categories that corresponds to some syntactic feature such as a noun phrase or a prepositional phrase.
  - An *interchunk* module which performs more global operations with the chunks and between them. More than one *interchunk* module can be used in sequence.
  - A *postchunk* module which performs finishing operations on each chunk and removes chunk encapsulations so that a plain sequence of lexical forms is generated.

  Each of the modules reads rules from files written in XML.
- A *morphological generator* which delivers a TL surface form for each TL lexical form, by suitably inflecting it. It reads a FST generated from a TL MD in XML.
- A *post-generator* which performs orthographic operations such as contractions (e.g. Spanish *del = de + el*) and apostrophations (e.g. Catalan *l'institut = el + institut*), using a FST generated from a rule file written in XML.
- A *reformatter* which de-encapsulates any format information.

3. Resources

As mentioned in the previous section, creating a machine translation system in the Apertium platform requires creating or adapting linguistic resources. As a consequence, for each of the 21 language pairs available there is at least: a finite-state morphology for analysis, another one for generation, a trained HMM-based part-of-speech tagger, a bilingual transfer lexicon,[3] and a set of transfer rules.

We describe below the current status of these resources for the platform as a whole, focussing on those resources which are stable (tested and proven). Apertium includes, in the words of Streiter et al. (2007), a *pool* of free resources for natural language processing targeted specifically at machine translation.

[3] A bilingual transfer lexicon contains correspondences between lemmas, parts-of-speech and in some cases between other morphological features.

3.1. Format filters

Format filters can be used also by other MT applications. The encapsulation of formatting is simple and eases the processing of multiple document formats in an efficient manner. The format filters available in Apertium include ODT, HTML, RTF, MediaWiki and others. Format descriptions are based on a simple XML specification.

3.2. Morphological dictionaries

The morphological transducers used in Apertium are built using the *lttoolbox* finite-state toolkit (Ortiz-Rojas et al., 2005). The toolkit provides: a compiler, to transform the dictionaries described in XML into the fast, compact finite-state transducers that are then used by the engine.

Morphological dictionaries (MDs) are written in a format (see Forcada et al. (2007) for details) that allows users to encode regularities in the form of paradigms that may in turn call other paradigms. The compiler takes advantage of this and builds the finite-state transducer recursively, performing local minimization at each step (Ortiz-Rojas et al., 2005).

It is worth noting during the discussion of MDs that there are many languages covered where the morphology in Apertium does not provide the widest coverage for a given language. This is certainly the case for English and Spanish. However, they are included as the uniform nature of the formats and tagsets can facilitate performing experiments, and the single licence (the GNU General Public Licence[4] (GPL) used throughout) ease their integration with other free software.

Table 1 gives a breakdown of the MDs currently available and some statistics of coverage. Some of these have been built from existing resources such as the the *Norsk Ordbank* (`http://www.edd.uio.no/prosjekt/ordbanken/`), *Eurfa* (`http://kevindonnelly.org.uk/eurfa/`), *Gramadóir* (`http://borel.slu.edu/gramadoir/`), or *Matxin* (`http://matxin.sf.net`). Numbers of lemmata are approximate and include multi-word units encoded in the lexicon, the lemmata of surface forms with attached clitics and, in some cases, duplicate entries for differing orthographies.

The surface column gives the total number of surface forms recognised by the analyser. The *mean ambig.* column gives the mean ambiguity for each surface form, that is the mean number of lexical forms (analyses) returned per surface form. This gives an indication of the completeness of the morphology, although in the case of languages with prefix inflection, such as Afrikaans and Persian, the dictionary may recognise surface forms that will never appear in running texts (overanalysis).

The coverage column gives *naïve coverage*, that is, the fraction of surface forms in a representative corpus for which at least one analysis is returned. The list of analyses returned may not be complete, hence the word *naïve*. Finally the corpus column gives

[4] `http://www.fsf.org/copyleft/gpl.html`

Language	Lemmata	Surface	Mean ambig.	Coverage	Corpus
N. Nynorsk[1] (nn)	47,193	402,096	1.33	89.6%	WP 2009-01-19
N. Bokmål[1] (nb)	46,945	571,411	1.30	88.2%	WP 2009-01-08
English (en)	33,033	75,761	1.23	95.2%	EP 2007-09-28
Afrikaans (af)	14,033	42,107	1.25	80.0%	WP 2009-07-31
Danish (da)	10,659	80,106	1.15	86.2%	EP 2007-09-28
Icelandic (is)	7029	206,353	2.41	82.0%	WP 2008-03-20
Swedish (sv)	5,130	37,191	1.08	80.0%	EP 2007-09-28
Asturian (ast)	46,550	13,549,353	1.16	86.3%	WP 2009-11-17
Spanish (es)	41,735	4,600,370	1.40	97.6%	EP 2007-09-28
Catalan (ca)	37,635	7,185,455	1.15	89.8%	WP 2009-10-10
French (fr)	28,691	275,007	1.32	95.6%	EP 2007-09-28
Galician (gl)	21,298	9,764,319	1.30	86.6%	WP 2009-02-01
Romanian (ro)	18,719	612,511	1.28	83.6%	WP 2009-11-23
Occitan (oc)	18,079	6,084,575	1.05	81.0%	WP 2009-11-23
Portuguese (pt)	11,156	9,330,910	1.78	94.9%	EP 2007-09-28
Italian (it)	10,117	462,319	1.25	88.8%	EP 2007-09-28
Breton (br)	13,999	278,279	1.10	87.6%	WP 2009-11-11
Welsh[2] (cy)	11,081	438,856	1.21	86.1%	WP 2009-11-10
Irish[3] (ga)	8,769	165,787	1.53	83.6%	WP 2009-11-08
Persian (fa)	11,087	514,539	1.06	80.0%	WP 2007-11-02
Bulgarian (bg)	14,413	169,121	1.11	80.5%	WP 2008-05-25

1. From *Norsk Ordbank* 2. From *Eurfa* 3. From *An Gramadóir* 4. From *Matxin*

Table 1: Statistics on Apertium finite-state morphological dictionaries organised by language family

details of the corpus on which the evaluation was performed, WP stands for Wikipedia and is followed by the date of the database dump,[5] EP stands for EuroParl (Koehn, 2005) and is followed by the release date. These corpora were chosen as they are available under free licences and are widely used in machine translation.

3.3. Bilingual lexica

Along with morphological analysers, Apertium also has a number of bilingual lexica. These are encoded in the same XML-based format used by the morphological analysers, but represent correspondences between lemmata, including multi-word units, parts of speech and, in some cases, morphological information (e.g. to specify

[5] http://download.wikipedia.org/

Pair	Entries	Pair	Entries	Pair	Entries	Pair	Entries
fr–ca	10,554	es–ca	40,446	en–gl	31,286	en–es	27,540
en–ca	24,601	fr–es	23,295	es–ro	21,511	oc–ca	18,896
es–it	17,294	oc–es	15,772	br–fr	15,762	es–ast	13,778
eu–es	12,174	pt–gl	11,844	es–pt	11,447	cy–en	11,405
sv–da	11,398	es–gl	10,807	pt–ca	7,716	is–en	5,875
nn–nb	73,809	ga–gd	7,863				

Table 2: Statistics on bilingual lexica available in Apertium as of November 11, 2009 (ISO-639 codes in Table 1; ga: Irish, gd: Scottish Gaelic)

changes in the inflection information from SL to TL, and also to mark some ambiguities that should be solved by the structural transfer module).

A summary of the available bilingual lexica in Apertium can be found in table 2. Included are dictionaries which are either in released language pairs, or otherwise considered reasonably stable.

3.4. Part-of-speech taggers

Apertium uses a first-order (bigram) HMM-based POS tagger (Cutting et al., 1992) that is trained from corpora and a tagger definition file (see below). It can be trained using classical methods —either supervised or unsupervised (Baum-Welch algorithm)— or by means of a novel unsupervised approach that uses the rest of the MT engine and a TL model to estimate the HMM parameters (Sánchez-Martínez et al., 2008).[6] An XML-based tagger definition file is used to specify how the lexical forms delivered by the morphological analyser must be grouped into coarse tags. Grouping lexical forms (consisting of a lemma and morphological information making up a rather "fine-grained" PoS tag) into coarse PoS tags is needed to reduce the amount of parameters of the HMM. Each coarse tag is defined by means of a list of fine-grained tags in which wild-cards can be used. Lexicalised coarse tags (Pla and Molina, 2004) may be defined where needed by specifying the lemma of the word in the corresponding attribute. HMM observable outputs are all the possible *ambiguity classes*, or sets of coarse tags occurring in the dictionary, plus a reasonable *open set* for unknown words.

It is also possible to define constraint rules in the form of *forbid* and *enforce* rules. *Forbid* rules define restrictions as sequences of two coarse tags that cannot occur. *Enforce* rules are used to specify the set of coarse tags allowed to occur after a particular coarse tag. These rules are applied to the HMM parameters by introducing quasi-zeroes in the state transition probabilities of forbidden sequences and re-normalising.

[6] A free/open-source implementation is provided by package apertium-tagger-training-tools.

3.5. Transfer rules

Transfer rules for each of the three transfer stages, *chunker*, *interchunk*, and *postchunk* are written using a very similar syntax. The rules are based on finite-state pattern matching and are non-recursive. They are largely hand-written (but see 4.2). *Chunker* rules deal with local phenomena such as number and gender agreement in noun phrases, local word reorderings, some lexical changes (e.g. of prepositions). *Interchunk* rules are used for analogous longer-range phenomena (such as the reordering of complete chunks) and can also be used to merge chunks; *Postchunk* may be used for internal adjustments after application of *interchunk* rules.

The average number of rules per direction in a language pair with multi-stage transfer is approximately 300, and in single stage transfer around 100. For example, the Spanish to Catalan direction has 104 single-stage rules, where the English to Catalan direction has 227 *chunker* rules, 59 *interchunk* rules and 38 *postchunk* rules.

4. Reuse and recycle

This section gives a review of ways in which the resources available in Apertium can be re-used in other MT systems, for example those based on the Moses (Koehn et al., 2007) statistical MT system, and how other machine translation systems can be used to create or improve resources for Apertium.

4.1. Reuse of resources in other systems

As described in Tyers (2009), the dictionaries of Apertium (sections 3.2 and 3.3), together with very basic transfer rules can be used to create full-form bilingual vocabulary lists which can be added to an existing parallel corpus for training a statistical machine translation system based on Moses. The idea of this list is to improve coverage of word forms for inflected languages, when using a small corpus, or when the corpus is of a limited domain (for example generating second-person singular forms of verbs where the corpus contains overwhelmingly third-person singular).

Adding the dictionary also eases the computation of accurate word alignments since one-to-one word mappings are explicitly provided. In Sánchez-Martínez and Forcada (2009), table 4 (p. 22) results are given for an SMT system trained on a small corpus when the generated bilingual corpus is added and when it is not.

4.2. Corpus-based creation and improvement of resources

A corpus-based approach to infer shallow structural transfer rules is proposed by Sánchez-Martínez and Forcada (2009).[7] The authors extend the alignment template

[7] A free implementation is provided by package `apertium-transfer-tools`.

approach (Och and Ney, 2004) used in statistical MT with a set of restrictions derived from the bilingual dictionary of Apertium to control their application as transfer rules. For the translation between closely-related languages, the authors report an improvement over word-for-word translation and a translation quality close to the one provided by hand-coded transfer rules. Their approach also provides better translation results than the Moses statistical MT system trained on the same small parallel corpus when it is extended with the Apertium bilingual dictionary (see Section 4.1).

It is worth noting that there has been another approach to the inference of shallow structural transfer rules using corpora and Apertium resources (Caseli et al., 2006) which, in addition to transfer rules, also automatically infers Apertium bilingual lexica.[8]

4.3. Hybridisation of Apertium and other machine translation systems

Sánchez-Martínez et al. (2009) have tested the integration of sub-sentential translation units (bilingual chunks) into the Apertium MT engine.[9] In their approach the bilingual chunks were automatically obtained from parallel corpora by using the marker-based chunkers and sub-sentential aligners used in the example-based MT system MaTrEx (Gough and Way, 2004; Tinsley et al., 2008).[10] Note, however, that bilingual chunks obtained in a different way could have been used, for instance the chunks[11] extraction algorithm (Zens et al., 2002) used by state-of-the-art statistical MT systems such as Moses.

In the integration of bilingual chunks into a rule-based MT system like Apertium, special care must be taken so as not to break the application of structural transfer rules, since this would increase the number of ungrammatical translations. Thanks to the modular design of Apertium this has been possible by developing a wrapper around the translation engine. The approach consists of (i) the application of a dynamic-programming algorithm to compute the best translation coverage of the input sentence given the collection of bilingual chunks available; (ii) the translation of the input sentence as usual by Apertium; and (iii) the application of a language model to choose one of the possible translations for each of the bilingual chunks detected. Sánchez-Martínez et al. (2009) report improvements, although not statistically significant, in the translation from English to Spanish, and vice versa.

[8] A free/open-source implementation can be downloaded from `http://retratos.sf.net`.

[9] A free/open-source implementation is provided by package `apertium-chunks-mixer`.

[10] Selected components from MaTrEx will soon be made available as the free/open-source package *Marclator* at `http://www.computing.dcu.ie/~mforcada/marclator.html`.

[11] Usually referred as *phrases* by statistical MT practitioners.

5. Discussion

We have presented in this paper the resources available in the Apertium machine translation platform, and some possible uses of these resources in improving other MT systems, or creating hybrid systems. The resources we present are currently used in 21 released rule-based machine translation systems.[12] Future research is aimed at: expanding the number of languages covered by the linguistic resources, increasing the number of language pairs, implementing a module for lexical selection, integrating other free/open-source software, such as HFST[13] or *foma* (Huldén, 2009) for managing more complex morphologies, and the implementation of a module for deeper structural transfer. Improving integration with other free/open-source machine systems such as Moses, Cunei and Matxin is also a priority.

Acknowledgements: We thank the support of the Spanish Ministry of Science and Innovation through project TIN2009-14009-C02-01. Mikel L. Forcada thanks the support given by Science Foundation Ireland (SFI) through ETS Walton Award 07/W.1/I1802.

Bibliography

Armentano-Oller, Carme, Rafael C. Carrasco, Antonio M. Corbí-Bellot, Mikel L. Forcada, Mireia Ginestí-Rosell, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Gema Ramírez-Sánchez, Felipe Sánchez-Martínez, and Miriam A. Scalco. Open-source Portuguese–Spanish machine translation. In *Computational Processing of the Portuguese Language, Proceedings of the 7th International Workshop on Computational Processing of Written and Spoken Portuguese, PROPOR 2006*, volume 3960 of *Lecture Notes in Computer Science*, pages 50–59. Springer-Verlag, May 2006. ISBN 3-540-34045-9.

Caseli, H.M., M.G.V. Nunes, and M.L. Forcada. Automatic induction of bilingual resources from aligned parallel corpora: application to shallow-transfer machine translation. *Machine Translation*, 20(4):227–245, 2006.

Cutting, D., J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Third Conference on Applied Natural Language Processing. Association for Computational Linguistics. Proceedings of the Conference*, pages 133–140, Trento, Italy, 31 mar–3 apr. 1992.

Forcada, Mikel L., Boyan Ivanov Bonev, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Gema Ramírez-Sánchez, Felipe Sánchez-Martínez, Carme Armentano-Oller, Marco A. Montava, and Francis M. Tyers. Documentation of the open-source shallow-transfer machine translation platform Apertium. `http://xixona.dlsi.ua.es/~fran/apertium2-documentation.pdf`, May 2007.

Forcada, Mikel L., Francis M. Tyers, and Gema Ramírez-Sánchez. The free/open-source machine translation platform Apertium: Five years on. In *Proceedings of theFirst International Workshop on Free/Open-Source Rule-Based Machine Translation FreeRBMT'09*, pages 3–10, November 2009.

[12] A full list may be found on the front page of the Apertium Wiki (`http://wiki.apertium.org`).

[13] `http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/`

Gough, N. and A. Way. Robust large-scale EBMT with marker-based segmentation. In *Proceedings of the Tenth Conference on Theoretical and Methodological Issues in Machine Translation (TMI-04)*, pages 95–104, Baltimore, MD., 2004.

Huldén, Måns. Foma: a finite-state compiler and library. *EACL 2009*, pages 29–32, 2009.

Karlsson, F., A. Voutilainen, J. Heikkilä, and A. Anttila. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text. Natural Language Processing, No 4*. Mouton de Gruyter, Berlin and New York, 1995.

Koehn, Philipp. Europarl: A parallel corpus for statistical machine translation. *MT Summit 2005*, 2005.

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. Moses: Open source toolkit for statistical machine translation. *Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session, Prague, Czech Republic, June 2007*, 2007.

Och, F. J. and H. Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449, 2004.

Ortiz-Rojas, Sergio, Mikel L. Forcada, and Gema Ramírez-Sánchez. Construcción y minimización eficiente de transductores de letras a partir de diccionarios con paradigmas. *Procesamiento del Lenguaje Natural*, (35):51–57, 2005.

Pla, F. and A. Molina. Improving part-of-speech tagging using lexicalized HMMs. *Journal of Natural Language Engineering*, 10(2):167–189, June 2004.

Sánchez-Martínez, Felipe and Mikel L. Forcada. Inferring shallow-transfer machine translation rules from small parallel corpora. *Journal of Artificial Intelligence Research*, 34:605–635, 2009.

Sánchez-Martínez, Felipe, Juan Antonio Pérez-Ortiz, and Mikel L. Forcada. Using target-language information to train part-of-speech taggers for machine translation. *Machine Translation*, 22(1-2):29–66, 2008.

Sánchez-Martínez, Felipe, Mikel L. Forcada, and Andy Way. Hybrid rule-based – example-based MT: Feeding apertium with sub-sentential translation units. In *Proceedings of the 3rd Workshop on Example-Based Machine Translation*, pages 11–18, Dublin, Ireland, November 2009.

Streiter, Oliver, Kevin P. Scannell, and Mathias Stuflesser. Implementing NLP Projects for Non-Central Languages: Instructions for Funding Bodies, Strategies for Developers. *Machine Translation*, 20(4):267–289, 2007.

Tinsley, J., Y. Ma, S. Ozdowska, and A. Way. Matrex: the DCU MT system for WMT 2008. In *Proceedings of the Third Workshop on Statistical Machine Translation, ACL 2008*, pages 171–174, Columbus, OH., 2008.

Tyers, Francis M. Rule-based augmentation of training data in Breton–French statistical machine translation. In *Proceedings of the 13th Annual Conference of the European Association of Machine Translation, EAMT09*, pages 213–218, 2009.

Zens, R., F. J. Och, and H. Ney. Phrase-based statistical machine translation. In *KI 2002: Advances in Artificial Intelligence: Proceedings 25th Annual German Conference on AI*, volume 2479 of *Lecture Notes in Computer Science*, pages 18–32. Springer-Verlag, 2002.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 77–86

Combining Content-Based and URL-Based Heuristics to Harvest Aligned Bitexts from Multilingual Sites with Bitextor

Miquel Esplà-Gomis[a], Mikel L Forcada[a b]

[a] Grup Transducens, Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, E-03071 Alacant, Spain
[b] Centre for Next Generation Localisation, Dublin City University, Dublin 9, Ireland

Abstract

Nowadays, many websites in the Internet are multilingual and may be considered sources of parallel corpora. In this paper we will describe the free/open-source tool Bitextor, created to harvest aligned bitexts from these multilingual websites, which may be used to train corpus-based machine translation systems. This tool uses the work developed in previous approaches with modifications and improvements in order to obtain a tool as adaptable as possible to make it easier to process any kind of websites and work with any pairs of languages. Content-based and URL-based heuristics and algorithms applied to identify and align the parallel web pages in a website will be described and, finally, some results will be presented to show the functionality of the application and set the future work lines for this project.

1. Introduction and background

Nowadays the biggest and most heterogeneous text corpus in the world is the World Wide Web. In fact, during the last years there have been many approaches to profit from the web as a corpus and, especially, as a text corpus. In our case, our approach is focused on using the web as a source of *bitexts* (parallel texts). It is known that many websites are, totally or partially, available in more than one language. This means that some of their web pages can be paired into bitexts.

Currently, bitexts have become a very important source of knowledge for the machine translation. It is in the area of corpus-based machine translation where the bitexts are more important. *Example-based* machine translation (EBMT) and *statistical* machine translation (SMT) need this kind of resources, for the process of training (Hutchins and Somers, 1992). In fact, there are corpora which have been obtained

Corresponding author: miquel.espla@ua.es
Cite as: Miquel Esplà-Gomis, Mikel L Forcada. Combining Content-Based and URL-Based Heuristics to Harvest Aligned Bitexts from Multilingual Sites with Bitextor. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 77–86. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0003-9.

from the Internet with the aim of training SMT, such as the Europarl Corpus (Koehn, 2005). There are even approaches to extract translation rules from parallel corpora used to create *rule-based* machine translation (RBMT) systems (Caseli and Nunes, 2007; Sánchez-Martínez and Forcada, 2009).

Based on this idea, different systems have been developed to harvest bitexts from the Internet. One of the earliest approaches is the STRAND system (Resnik and Smith, 2003), which is designed to identify web pages which are candidates to be bitexts. This system uses the HTML structure and the text-block length to compare files between them through the application of different calculations and thresholds. Similar approaches have been developed with this kind of methods to harvest bitext from the web (Chen and Nie, 2000; Kit et al., 2005; Désilets et al., 2008). In these cases, the system used to obtain the preliminary candidates for each web page is the identification and substitution of language markers in the URLs (Nie et al., 1999) (this will be covered in the section 3). In our approach we have not used this system in order to create an application as independent as possible of the website structure and the language pair searched.

Taking all these ideas, Bitextor was created as a free/open-source tool with the aim of obtaining the maximum number of parallel texts from multilingual websites, aligning them and generating translation memories (TMs) in TMX format.[1] To do this, the content comparison techniques developed in the cited projects have been applied with some modifications, combining them with other heuristics which will be explained in next sections. To assist in this task, another free/open-source application has been used: the TagAligner tool (Sanchez-Villamil et al., 2006), which both uses the tag structure in XML files and the length of the sentences in a pair of documents to align them (Brown et al., 1991; Gale and Church, 1994).

2. Obtaining and preprocessing web pages

To start the process of obtaining TM from a multilingual website with Bitextor, the first step is to download the entire website. To do this, Bitextor uses the tool HTTrack,[2] which is able to filter and download only the HTML files in the website. All these files are saved locally and are tagged with their URL.

Once this is done, some normalisation tasks are performed on the files in order to convert them into a valid format for processing. Firstly, Bitextor uses the library LibEnca[3] to detect the original character set encoding. It then uses LibTidy[4] to normalize the HTML files into valid XHTML files and to convert the detected original encoding into UTF-8.

[1] `http://www.lisa.org/Translation-Memory-e.34.0.html` [Last visited: 26th November 2009]

[2] `http://www.httrack.com` [Last visited: 26th November 2009]

[3] `http://sourceforge.net/projects/freshmeat_enca/` [Last visited: 26th November 2009]

[4] `http://tidy.sourceforge.net` [Last visited: 26th November 2009]

3. Choosing the parameters for the comparison

To look for web page pairs that are bitexts, Bitextor needs to obtain and save some features from these pages. In this section, we will explain how this is done.

3.1. Surface features

The first features observed in a web page are those which we will consider, in this paper, *surface features*. These are the features that may not be used for an accurate comparison, but can be used as a indicator to discard very unlikely pairs of files. In our approach, these features are:

- *Text-language comparison*: It is obvious that if two files are written in the same language, one of them can not be a translation of the other one. The language in which each text has been written is detected and stored by using LibTextCat.[5]
- *File size ratio*: This parameter is relative and is used to filter pairs of files whose size is very different.
- *Total text length difference*: This parameter has the same function that the previous one, but compares the size of each file's plain text in characters.

3.2. Web page content

In order to obtain a more precise comparison Bitextor uses web page content in the comparison process. Web pages have an advantage over plain text: they are tagged with format and structure tags, which provide additional information that can be used to compare them. The idea is that two parallel web pages should have the same HTML tag structure (or, at least, a similar one).

Basically, two elements in the content of the web pages are considered in our approach: the HTML tag structure and the text block length. This is the same information used in the STRAND approach. In Bitextor, the extraction of this information is divided into two steps: the file cleaning and the encoding. In the first one, the objective is to remove all the irrelevant information, such as comments, the heading of the web page, the tag parameters, the irrelevant HTML tags and the extra spaces in the text blocks. In the second step Bitextor encodes the remaining information into a string in which two kinds of information can be represented: the tag names and the text block lengths (measured in characters[6]). This string acts as a *fingerprint* of the web page.[7] We can see an example of this kind of encoding in the Figure 1. This method of encoding provides the possibility of using the edit-distance algorithm to make the comparison.

[5] `http://software.wise-guys.nl/libtextcat/` [Last visited: 26th November 2009]

[6] An interesting study issue could be to analyse the differences in the results calculating the length of the text blocks in characters or in words

[7] To optimise calculations, this information is encoded with integers.

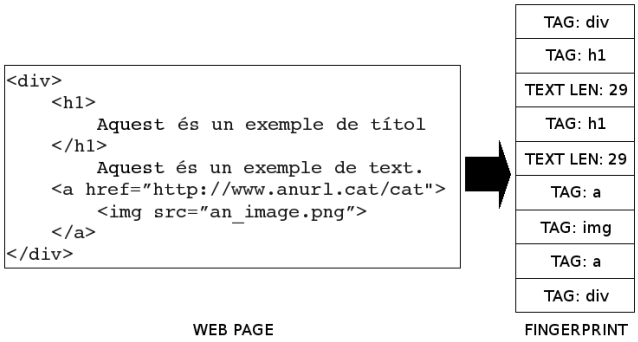

Figure 1. Example of conversion from HTML to a fingerprint string.

3.3. URL

One difference between this work and other previous approaches is that Bitextor does not download the candidate files by using rules of detection and substitution of language markers in the URL (Nie et al., 1999). Bitextor downloads the whole website and, then, uses the URLs as one more parameter to discard pairs of files with low probability to be bitexts. In order to do this, Bitextor divides the URL of a file into three sections: thedirectory path, the filename and the variables. In this way, it can compare each section separately.

4. Web page comparison process

To compare the web pages, the features explained in the previous section are compared one by one. Firstly, the surface features are compared in order to discard the most obviously incorrect pairs of files. With the remaining files the following two methods are applied.

4.1. URLs comparison

For the URL comparison Bitextor applies a restriction: candidate pairs can have at most, one difference in their URL. In practice, this implies one of these three possibilities in which the difference can be:

The filename: This is the simplest difference. When both files are saved in the same directory but they have a different name:

http://www.gnu.org/home.ca.html ⇒ http://www.gnu.org/home.en.html

A directory: This means that the directory structure differs in the name of a directory. For example it happens when files are saved in a path with the same structure but that is forked in a particular level in the directory tree. This can also happen when one of the files is saved in a subdirectory of the directory where the other one is saved:

http://www.ua.es/va/index.html ⇒ http://www.ua.es/en/index.html

http://www.ua.es/index.html ⇒ http://www.ua.es/en/index.html

A variable: This difference consists in the fact that the same file is called using a variable with a different value in each of the cases. It can also happen when one file has one more variable:

http://www.dlsi.ua.es/index.html?id=val ⇒ http://www.dlsi.ua.es/index.html?id=eng

http://www.dlsi.ua.es/index.html ⇒ http://www.dlsi.ua.es/index.html?id=eng

With this system, Bitextor tries to take advantage of the information provided by the URL without having to manually generate the rules of pattern recognition and substitution of language markers in the web pages URL. It means that Bitextor can be used directly on any website without having to analyse its structure.

4.2. Web page content comparison

Finally, those pairs that have not been discarded in the previous step are compared through their fingerprint (see section 3.2). To do this, Bitextor uses the Levenshtein edit distance algorithm (Levenshtein, 1966). It is important to explain that, when Bitextor applies this algorithm on the obtained fingerprints, it has to process two kinds of elements (tags and text blocks). The comparison between XHTML tags is simple: they can be different or equal. However, the comparison between text blocks is not as easy. As in other methods (Gale and Church, 1994), the length is the parameter used to perform the comparison between text blocks, so the most reasonable option seems to be to use the following measure of divergence between the length two text block lengths b_1 and b_2:

$$D(b_1, b_2) = \frac{|b_1 - b_2|}{\max(b_1, b_2)} \quad (1)$$

In fact, in our approach we implement two ways to use this information in order to obtain two different values as a result of the edit distance calculation. The first way is to set a threshold for $D(b_1, b_2)$ for each pair of languages. In this way, we can evaluate if two texts blocks could be parallel or not. The value obtained from applying the edit-distance with this method is used to discard improbable pairs by defining a maximum number of absolute differences between both fingerprints. The other way

to compare the text block lengths is to directly use $D(b_1, b_2)$ as a cost. The value obtained from the edit-distance calculation with this method is used to know which is the most probable candidate for a given file from the group of files that may have passed all the other heuristics (the one having the lowest value).

In this way, for the operations defined for the Levenshtein edit-distance (insertion, deletion and substitution) we can define the following cost functions: for insertions $C_i(x)$ and deletions $C_d(x)$, the cost is the same for tags and a text block lengths, independently of its length x:

$$C_i(x) = 1 \qquad C_d(x) = 1 \tag{2}$$

For substitutions of tags (t) we will have the cost function $C_s(t_1, t_2)$:

$$C_s(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 1 & \text{if } t_1 \neq t_2 \end{cases} \tag{3}$$

In the case of text block lengths b_1 and b_2, as we have said, we have two functions: the direct cost function without using the threshold $C_s(b_1, b_2)$ and the cost function using the threshold $C'_s(b_1, b_2)$:

$$C_s(b_1, b_2) = D(b_1, b_2) \qquad C'_s(b_1, b_2) = \begin{cases} 1 & \text{if } D(b_1, b_2) > t_b \\ 0 & \text{if } D(b_1, b_2) \leq t_b \end{cases} \tag{4}$$

Substitutions between tags and text block lengths are not allowed:[8]

$$C_s(t_1, b_1) \to \infty \qquad C_s(b_1, t_1) \to \infty \tag{5}$$

5. Aligning the obtained websites

The last task performed by Bitextor is the alignment of the candidates. In order to align a pair of XHTML files, Bitextor uses the LibTagAligner to perform the alignment. The method used by TagAligner to align files is similar to the one used by Bitextor to compare them. TagAligner encodes the file with a fingerprint (as Bitextor does), but it uses a more detailed weight structure with the edit-distance algorithm. In contrast to Bitextor, when this algorithm is performed, not all tags are compared in the same way. This tool allows the user to group the tags in categories. For these categories, the user can define weights for the operations defined in the edit-distance algorithms.

Thus, for a tag t in a category k, the cost of an insertion $C_i(t)$ or deletion $C_d(t)$ operation is expressed by the functions:

$$C_i(k) = W_i(t) \qquad C_d(t) = W_d(k) \tag{6}$$

[8] In our approach, we assign the C++ `MAXDOUBLE` value to these cost functions.

where $W_i(k)$ and $W_d(k)$ are the functions that return the weights assigned by the user for the insertion and deletion operations.

In the case of substitution, the cost function for two tags (t_1 and t_2) in two categories (k_1 and k_2) is:

$$C_s(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ W_s(k_1, k_2) & \text{if } t_1 \neq t_2 \end{cases} \quad (7)$$

where $W_s(k_1, k_2)$ is the function that determines the cost of a substitution on a pair of different tags belonging to the same category or two different categories.[9]

Weights are also assigned to text block length operations, and they are relative to the length of the blocks operated. But, in contrast to the fingerprint comparison used by Bitextor, in LibTagAligner the user can choose whether the length of the text block is measured in characters or words. So, in our case, the set of cost functions for text blocks b is:

$$C_i(b) = W_i(b) \cdot b \qquad C_d(b) = W_d(b) \cdot b \quad (8)$$

$$C_s(b_1, b_2) = W_s(b_1, b_2) \cdot |b_1 - b_2| \quad (9)$$

Again, tag–text block substitutions are not allowed, so, the cost of the operation will be implemented as an infinite (as has been explained in section 4.2).

6. Results

This section presents results from the system. These tests have been performed by using version 3.2.0 of Bitextor.[10] What we are going to analyse is the capacity of Bitextor to find the parallel web pages in a given website. In terms of alignment quality, there is a complete study (Sanchez-Villamil et al., 2006) with results about this issue. The metrics used to evaluate Bitextor have been precision and recall. We define precision (P) as the number of correct pairs obtained (N_C) over the total number of pairs obtained (N_T). The recall (R) is then the number of correct pairs obtained (N_C) over the total number of possible pairs in the website (N):

$$P = \frac{N_C}{N_T} \qquad R = \frac{N_C}{N} \quad (10)$$

It is obvious that it would be a huge work to check all the pairs of files generated by Bitextor, or find the total number of possible pairs of files in a website composed of thousands of web pages. To obtain an approximate estimation of the precision, we have randomly obtained a sample of 100 pairs generated by Bitextor and have checked them by hand. In the same way, we have obtained a list of 300 web pages

[9] An optimal set of weights can be found in (Sanchez-Villamil et al., 2006).

[10] The configuration file used to perform the tests can be looked up in the trunk of the SVN server of Bitextor for its revision 146: `https://bitextor.svn.sourceforge.net/svnroot/bitextor/trunk/`

from the downloaded website and have tried to find them in the list of pairs generated by Bitextor. Then, have checked if these pairs were correct or not.

For a first test, we have tried with a very simple case: the website of the Parliament of Canada.[11] This country has two official languages (English and French), and this website must have all its pages in the both languages, so, in theory, all the pages must have a bitext candidate. The website was downloaded by using HTTrack and 56,173 HTML files were obtained. Bitextor was applied with a threshold of 10 maximum differences between fingerprints. From the website, 24,717 pairs of web pages were found. The results in this case were very satisfactory: P=99% and R=85,33%.

These are very promising results, but, probably, the quality of the extraction is probably due to the fact that this is a very well structured website, with uniform language markers in the URL and highly parallel contents. Because of this, we wanted to try with a more complex case. The next results were obtained from a website from the Universitat d'Alacant,[12] which is written in three languages: English, Catalan and Spanish. This website is heterogeneous, with some pages without any translation and with multiple systems to mark the language in the URL. In this way, these were the obtained results: $P = 86\%$ and $R = 61\%$. Obviously, these results are worse than the obtained in the previous test. There are some reasons to explain what has happened in this case with the precision: the noise caused by web pages without any translation, the fact that some pages have no language marker in the URL, the presence of pages with a very similar content and the same structure (for example, the staff section, which uses a template for all the web pages and only changes a few lines of text).

Analysing the results one by one, we have noticed that, in many occasions it is better to have a lower recall because in many of the discarded pairs, the information contained in the pages was minimal (only some words), with mixed languages, only numeric data, etc. So, it is important to understand that the recall can be more related with the quality of the website as a parallel corpus than with the performance of the application.

7. Conclusions

After this study, we can extract some conclusions. Firstly, it is clear that this system gives promising results for webpages with a high number of parallel pages and with not much noise. Certainly, it is probable that many of the multilingual webpages in the Internet do not fit this profile. Thus, it seems that one of the most important future lines of work in this project will be to develop new heuristics to clean all the possibly noisy files.

Regarding a comparison of previous works in the area and Bitextor, we have obtained some good results, comparable to those obtained with other similar approaches

[11] `http://www.parl.gc.ca`

[12] That of the Department of Computer Languages and Systems, `http://www.dlsi.ua.es`

(although it is difficult to quantify without applying them on the same websites in a controlled work environment). In addition, the system of comparison of URLs of Bitextor has been designed to be more adaptable and, as consequence, obtain better results for any website without studying its structure.

One important point in our approach is the fact that it is a free/open-source tool. We think that free/open-source is very important in this kind of applications, since we are working with a very heterogeneous material: websites are very different between them, different corpora with different languages can present very different problems (for example, the alignment), etc. With a free application we are allowing people to try our application and to add new features to face all the possible problems.

8. Where to find Bitextor and TagAligner

Bitextor and LibTagAligner are under the GNU General Public License (GPL) version 2.0[13] and they are available for UNIX-like platforms. Its code and releases can be found at `http://sourceforge.net/projects/bitextor` and `http://sourceforge.net/projects/tag-aligner`.

9. Future work

Currently, there are various tasks pending for the Bitextor project. We are exploring ways to increase the precision of our system in order to obtain better results on noisy websites. Another important task planned is the integration of Bitextor with other free tools, like Bitext2TMX[14] to create a more powerful work environment for the creation and editing of translation memories.

Another important improvement would be to add a new module to allow Bitextor to acquire by itself candidate websites to be parallel (for a given pair of languages) (Leturia et al., 2009).

Acknowledgements: The original development of Bitextor was funded by the Ministerio de Ciencia y Tecnología (Spanish Government) between 2004 and 2006 through grant (TIC2003-08681-C02). Later, it was funded by the Universitat d'Alacant. Enrique Sánchez Villamil was the author of the 1.0 version of Bitextor and the 1.0 version of TagAligner (on which the initial version of LibTagAligner, used by Bitextor was based). Miquel Simón i Martínez was the author of the second version (v2.0) of TagAligner, with an improvement in configurability through the incorporation of an XML configuration file. MLF's stay at Dublin City University is funded by Science Foundation Ireland through an ETS Walton Award. We thank the support of the

[13] `http://www.gnu.org/licenses/gpl-2.0.html` [Last visited: 26th November 2009]

[14] `http://www.sf.net/projects/bitext2tmx`

Spanish Ministry of Science and Innovation through project TIN2009-14009-C02-01. We thank Francis M. Tyers for comments on the manuscript.

Bibliography

Brown, P.F., J.C. Lai, and R.L. Mercer. Aligning sentences in parallel corpora. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 169–176. Association for Computational Linguistics Morristown, NJ, USA, 1991.

Caseli, HM and MGV Nunes. Automatic induction of bilingual lexicons for machine translation. *Int J Transl*, 19:29–43, 2007.

Chen, J. and J.Y. Nie. Parallel web text mining for cross-language IR. In *Proceedings of RIAO 2000: Content-Based Multimedia Information Access*, volume 1, pages 62–78, 2000.

Désilets, A., B. Farley, M. Stojanovic, and G. Patenaude. WeBiText: Building Large Heterogeneous Translation Memories from Parallel Web Content. *Proc. of Translating and the Computer*, 30:27–28, 2008.

Gale, W.A. and K.W. Church. A program for aligning sentences in bilingual corpora. *Computational linguistics*, 19(1):75–102, 1994.

Hutchins, W.J. and H.L. Somers. *An introduction to machine translation*. Academic Press New York, 1992.

Kit, Chunyu, Xiaoyue Liu, KingKui Sin, and Jonathan J. Webster. Harvesting the bitexts of the laws of Hong Kong from the Web. 2005.

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

Leturia, I., I. San Vicente, and X. Saralegi. Search engine based approaches for collecting domain-specific Basque-English comparable corpora from the Internet. In *Proceedings of the 5th Web As a Corpus, Sociedad Española de Procesamiento del Lenguaje Natural Conference*, 2009.

Levenshtein, V.I. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

Nie, J.Y., M. Simard, P. Isabelle, and R. Durand. Cross-Language Information Retrieval based on Parallel Texts and Automatic Mining of Parallel Texts from the Web. In *Proceedings of SIGIR'99: 22nd International Conference on Research and Development in Information Retrieval: University of California, Berkeley, August 1999*, page 74. Association for Computing Machinery (ACM), 1999.

Resnik, P. and N.A. Smith. The web as a parallel corpus. *Computational Linguistics*, 29(3): 349–380, 2003.

Sánchez-Martínez, Felipe and Mikel L. Forcada. Inferring shallow-transfer machine translation rules from small parallel corpora. *Journal of Artificial Intelligence Research*, 34:605–635, 2009.

Sanchez-Villamil, E., S. Santos-Anton, S. Ortiz-Rojas, and M.L. Forcada. Evaluation of alignment methods for HTML parallel text. *Lecture Notes in Computer Science*, 4139:280, 2006.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 87–96

Fast and Extensible Phrase Scoring for Statistical Machine Translation

Christian Hardmeier

Fondazione Bruno Kessler, Trento, Italy

Abstract

Existing tools for generating phrase tables for phrase-based Statistical Machine Translation (SMT) are generally optimised towards low memory use to allow processing of large corpora with limited memory. Whilst being a reasonable design choice, this approach does not make optimal use of resources when the sufficient memory is available. We present *memscore*, a new open-source tool to score phrases in memory. Besides acting as a faster drop-in replacement for existing software, it implements a number of standard smoothing techniques and provides a platform for easy experimentation with new scoring methods.

1. Motivation

Phrase tables for Statistical Machine Translation (SMT) systems are commonly built from very large parallel corpora in order to obtain ample vocabulary coverage and sufficient quality of the translation probability estimates. On usual desktop computers, the size of the phrase tables extracted from a large corpus will often exceed the size of the physical memory available on the machine. Software tools used to estimate phrase tables from parallel corpora are designed to take this constraint into account. They do not try to load the complete data into memory at once. Instead, they process their inputs as data streams, relying on local information only, and make extensive use of temporary disk files and intermediate disk-bound sorting passes to access the right information at the right moment.

This approach ensures that the size of the parallel corpus that can be processed is limited only by the potentially very large amount of disk space available; the use of working memory is kept to a minimum. The result is almost unlimited scalability

Corresponding author: hardmeier@fbk.eu
Cite as: Christian Hardmeier. Fast and Extensible Phrase Scoring for Statistical Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 87–96. ISBN 978-80-904175-4-0.
doi: 10.2478v10108-010-0007-5.

to very large corpora given sufficient disk space, but it seems wasteful not to exploit the available memory resources as fully as possible. Moreover, the stream-based approach to data processing makes it very expensive to access data that is not locally available, so the scoring functions that can be implemented are essentially limited to those that require only a small number of straight passes through the data. In recent years, random-access memory has become much cheaper, and the general availability of 64-bit computers has lifted another important restriction on memory size, such that academic sites now have access to computing equipment which can handle data sets in memory that were beyond reach even a few years ago. It is reasonable to use software that takes advantage of these new capabilities, not only to speed up SMT system training, but also to make it feasible to implement new phrase scoring algorithms that require more than just a few passes through the data.

Phrase-based Statistical Machine Translation (SMT) uses translation models in the form of phrase tables, in which phrase pairs consisting of a source language (SL) and a target language (TL) word sequence, s and t, are associated with a number of scores corresponding to different models of translation probabilities between s and t. Following Koehn et al. (2003), candidate phrase pairs are usually extracted from a parallel corpus with automatically generated word alignments. The forward and reverse conditional phrase translation probabilities $p(s|t)$ and $p(t|s)$ are then estimated by the relative frequency of a SL phrase in alignment with a given TL phrase and vice versa. To overcome the unreliability of these estimates for low-frequency phrases, phrase tables usually include maximum likelihood scores for both $p(s|t)$ and $p(t|s)$ as well as two additional lexical weight scores based on the word alignment probabilities of the individual component words of the source and the target phrases (Koehn et al., 2003).

The widely used *moses* toolkit for Statistical Machine Translation (Koehn et al., 2007) includes a tool called *phrase-extract* to extract phrase pairs from a word-aligned corpus and compute phrase translation probabilities and lexical weights. It is designed to process large amounts of corpus data on computers with relatively little random-access memory (RAM). To achieve this, the file system is used extensively to store temporary data. Phrases pairs are extracted from the parallel corpus and stored to disk. Scoring is done individually for the two forms of the conditional probability, $p(s|t)$ and $p(t|s)$, and for each scoring pass, the extracted phrases have to be sorted by target or source phrase, respectively. After scoring, the output of one of the two scoring runs is sorted again to match the order of the other run for merging. Another open-source implementation of SMT phrase scoring, Thot (Ortiz-Martínez et al., 2005), also works with temporary disk files to cut down on RAM usage.

In this paper, we present *memscore*, an open-source phrase scoring tool replicating and extending the functionality of the *score* component of the *phrase-extract* software bundled with *moses*. Like *score*, it takes as input a list of phrase pairs produced by the *extract* tool of *phrase-extract* and calculates phrase translation scores. Unlike *score*, it performs all the computations in RAM and does not require the input to be sorted in any way. As a result, scoring is much faster for data sets that fit completely in memory.

p(s|t)	p(t|s)	
`-s ml`	`-r ml`	maximum likelihood score
`-s wittenbell`	`-r wittenbell`	Witten-Bell smoothing
`-s absdiscount`	`-r absdiscount`	absolute discounting
`-s lexweights <file>`	`-r lexweights <file>`	lexical weights
`-s const <constant>`		constant phrase penalty

Table 1. memscore *command line options*

Its implementation as a C++ program designed with modularity in mind makes it easy to experiment with different scoring techniques. A small number of smoothing techniques are already implemented, and other methods can easily be added. The framework has also been used successfully for an experimental implementation of an iterative scoring algorithm. In the rest of the paper, we are going to describe the typical usage of *memscore*, some implementation details useful to those who want to implement their own scoring algorithms in the framework provided by the tool and a comparison of *memscore* with the *phrase-extract* scorer in terms of runtime performance on two common SMT tasks.

2. Usage

2.1. Invocation

The *memscore* tool takes as input a list of phrases extracted from a parallel corpus in the format used by the *phrase-extract* tool bundled with the *moses* decoder. The phrase list is read from standard input and does not need to be sorted. It prints to standard output a phrase table in the format used by *moses*.

The scores to be included in the phrase table are specified on the command line with the switches listed in table 1. Each score is selected by one of the command line options `-s` or `-r` followed by the identifier of the scorer. Additional arguments may follow if the scorer requires this. When the option `-s` is used to specify a scorer producing a conditional probability, the probability $p(s|t)$ is generated. Using the scoring option `-r` requests that the inverse probability $p(t|s)$ be output instead. Thus, to produce a phrase table with maximum likelihood probabilities and lexical weights in both directions and a constant phrase penalty, as typically created by the *moses* training scripts, you would use the following command line:

```
gzip -cd model/extract.gz |
   memscore  -s ml -s lexweights model/lex.e2f \
             -r ml -r lexweights model/lex.f2e \
             -s const 2.718 |
   gzip >model/phrase-table.gz
```

Here, the files `lex.e2f` and `lex.f2e` contain the lexical translation tables generated by the *moses* training script in training step 4, and `extract.gz` is the phrase extraction file produced in step 5. The *memscore* command itself replaces the scoring step 6. We plan to integrate this step smoothly into the standard *moses* training script, but at the time of writing, this has not been done yet.

In the configuration mentioned in the previous paragraph, the output of *memscore* is essentially identical to that of the reference implementation, *phrase-extract*. The estimates of the maximum-likelihood scores are exactly the same as those produced by *phrase-extract*. The lexical weights can be different if a certain phrase pair occurs in the input with more than one set of alignments. According to Koehn et al. (2003), the maximum score generated by any of the alignments should be used in this case. However, the reference implementation does not conform to this recommendation. Instead, it computes the lexical weight based on the alignment with the highest count in the input. If there are several alignments with equal counts, the one occurring earliest in the input stream is selected. Thus, the actual choice depends on the sorting order of the input. In our implementation, two different modes of operation are available: By default, *memscore* outputs the maximum lexical weight as suggested by Koehn et al. (2003). If the command line switch of the lexical weight scorer is given as `-s lexweights -AlignmentCount model/lex.e2f`, the lexical weight is based on the most frequent alignment instead. If there is a tie for the maximal count, the greatest score generated by any of the competing alignments is chosen. This mode of calculation matches the *phrase-extract* scoring more closely, but differences are still possible in a small number of cases.

2.2. File formats

The file formats processed by *memscore* are the same as those produced and used by the *moses* toolkit. They are illustrated in table 2. As input, a list of phrase pairs extracted from a parallel corpus is read. The three fields in each line, separated by the characters '␣||␣', are the source phrase, the target phrase and the word alignment. The alignment is specified as a list of alignment links between word numbers, where, e.g., a link `0-1` indicates that the first source word is aligned to the second target word. Each phrase pair should occur in the input as often as it can be extracted from the corpus. Sorting is not required. A suitable file is produced by the *moses* training procedure under the name `extract.gz`. The inverse extraction file `extract.inv.gz` is not needed when *memscore* is used.

The phrase table produced by *memscore* uses the same field delimiters. After the source and target phrases, the word alignment is given in a different format. For each source word, the third field contains a pair of parentheses with a comma-separated list of word indices in the target phrase aligned to this word. In the fourth field, there is a similar list for each target word. When a phrase pair occurs with different alignments in the input, the most frequent alignment is output. Ties are broken arbitrarily. The

Phrase extraction file (input):
```
gemäß ||| in accordance with ||| 0-0 0-1 0-2
```
Phrase table (output):
```
gemäß ||| in accordance with ||| (0,1,2) ||| (0) (0) (0) ||| s1 s2 ...
```
Lexical translation table:
```
gemäß accordance 0.0445155
```

Table 2. File record formats used by memscore

fifth field of the phrase table record contains the scores $s_1, s_2, \ldots$ as floating point numbers in the order in which the scorers were specified on the command line.

The lexical weight scorer additionally requires a list of lexical translation table as input. This table has records with three blank-separated fields giving the source word, the target word and the lexical translation score, which is estimated as the number of alignments between the source word and the target word in the corpus, divided by the number of occurrences of the target word. When the lexical weight scorer is used in reverse mode, the word translation probabilities must also be reversed. These are the files `lex.e2f` and `lex.f2e` provided by the *moses* training scripts.

3. Implementation

3.1. Architecture

The architecture of *memscore* has been designed to favour extensibility. Developers should be able to implement quickly new scoring mechanisms without having to spend time on parsing input files, designing compact data structures and dealing with memory management. The scoring code is cleanly separated from these ancillary functions. Also, computing the forward and reverse conditional probabilities $p(s|t)$ and $p(t|s)$ is handled transparently by *memscore*. The programmer only has to provide one implementation for the form $p(s|t)$; reverse scoring is available automatically.

The main components of *memscore* are outlined in table 3. The classes `PhraseTable`, `PhraseInfo` and `PhrasePairInfo` provide access the data structures in which information about the phrase table is stored. The algorithmic components are encapsulated in the subclasses of `PhraseStatistic`, which can be used to compute statistics about individual source language and target language phrases, and those of `PhraseScorer`, which represent the actual scoring algorithms, respectively.

The class `MemoryPhraseTable` takes care of parsing the input data and storing it in memory using a hash table provided by the C++ standard template library. The source and target phrases are stored in hash tables of their own and represented internally by numeric identifiers. For each phrase pair, a `PhrasePairInfo` data structure encapsulates the joint counts. For each SL or TL phrase, a `PhraseInfo` structure

Parent class	Derived classes	Description
PhraseTable	MemoryPhraseTable ReversePhraseTable	provide access to the phrase table
PhraseInfo		stores data about single phrases
PhrasePairInfo		stores data about phrase pairs
PhraseStatistic	PhraseLanguageModel ClosedPhraseLanguageModel	compute phrase-level statistics
PhraseScorer	MLPhraseScorer WittenBellPhraseScorer AbsoluteDiscountPhraseScorer ConstantPhraseScorer	implement phrase scoring algorithms

Table 3. Principal components of memscore

contains the marginal counts and the number of distinct phrases of the other language it is aligned with. Both the `PhraseInfo` and the `PhrasePairInfo` classes also have a mechanism by which more sophisticated scoring algorithms can request additional storage to be associated with phrases or phrase pairs. If, e. g., a phrase language model is used, it will ask for space to be reserved to cache the language model scores for each phrase. To avoid excessive memory consumption by features that are not actually used, the extra information for phrase pairs is stored in a variable-sized data structure that only includes the information actually used by the scoring algorithms selected by the user in a particular run. For the implementor of a scoring algorithm, this is handled transparently.

The class `ReversePhraseTable` is an adapter that provides access to the phrase table with the source and the target side exchanged. This makes it possible to use exactly the same implementations of any scorer for computing both $p(s|t)$ and $p(t|s)$.

Subclasses of `PhraseStatistic` calculate statistics of single SL or TL phrases, which can then be used by the actual scoring algorithm. This feature is not used by the standard scorers, all of which estimate scores based on phrase counts alone. However, an experimental scorer might also take into account other characteristics of the phrases. We provide two implementations of this interface: The `PhraseLanguageModel` class scores the phrases with an IRSTLM language model (Federico et al., 2008). The `ClosedLanguageModel` does the same, but normalises the language model scores over the phrases encountered in the phrase table, assuming a closed world of phrases enumerated by the input.

Finally, the subclasses of `PhraseScorer` do the actual scoring. At the moment, three algorithms with very simple implementations are available (table 4). The `MLPhraseScorer` computes the standard maximum likelihood estimate for a multinomial distribution based on relative frequencies. The `WittenBellPhraseScorer` uses

$$p(s|t) = \frac{c(s,t)}{c(t)} \qquad p(s|t) = \frac{c(s,t)}{c(t) + N_s(t)} \qquad p(s|t) = \frac{c(s,t) - \beta}{c(t)}$$

maximum likelihood | Witten-Bell | absolute discounting

$c(\cdot)$: (joint or marginal) counts β: discounting constant (see text)
$N_s(t)$: number of distinct s occurring with t

Table 4. Scoring methods implemented in memscore

the Witten-Bell estimate known from language modelling (Witten and Bell, 1991). Another estimate borrowed from language modelling (Ney et al., 1994) is calculated by the `AbsoluteDiscountPhraseScorer`, which reduces the joint count of each event by a discounting constant $\beta = n_1/(n_1 + 2n_2)$, where n_1 and n_2 are the number of phrase pairs occurring exactly once or twice in the parallel corpus, respectively. In both cases, the probability mass is not redistributed to any backoff distributions, so the probabilities will not sum to 1 over the closed world of the phrase table. This type of smoothing avoids the typical overconfident estimates for phrase pairs with low counts that maximum likelihood estimation is subject to.

3.2. Memory management

The operation of *memscore* can be divided in three phases. First, the input data is loaded and the internal data structures are constructed. Next, the `PhraseScorer` classes have the opportunity to collect any statistics they require by accessing the data in an arbitrary way. The maximum likelihood and Witten-Bell scorers do nothing in this stage; the absolute discounting scorer computes its discounting constant β. Finally, the scorers are requested to emit their score estimates in a final, ordered pass through all the phrase pairs.

In terms of memory consumption, the first stage is characterised by the allocation of a very large amount of memory for a multitude of small objects representing phrase or phrase pair properties. In the next two phases, memory usage remains essentially constant; all the memory is freed at once on program termination. The default memory allocators do not cope well with this usage pattern. Memory and execution time profiling revealed that the excessive allocation of small objects leads to overhead memory consumption of up to one third of the total amount of space requested because the memory allocator associates a certain amount of accounting information with each memory block allocated, and the time wasted on memory allocation and deallocation far exceeds the time spent on scoring. To overcome these problems, considerable effort went into the optimisation of memory allocation patterns. In some important cases, allocation and freeing of large numbers of small objects was made considerably more efficient by judicious use of the memory pools provided by the

	Europarl DE-EN		**NIST** AR-EN	
Corpus size (sentences)	1,252,747		4,654,686	
Corpus size (English tokens)	34,731,010		147,135,694	
Phrase pairs (types)	28,251,755		115,474,492	
Phrase pairs (instances)	64,271,574		318,961,124	
	score	*memscore*	*score*	*memscore*
Time (h : mm)	1 : 05	0 : 26	5 : 46	2 : 14
Peak memory usage	12.3 GB	15.7 GB	5 GB	62.2 GB
Single iteration		13.2 s		38.6 s

Table 5. Scorer performance on Europarl and NIST tasks

Boost library, which allocate single large pools of memory to hold many small objects of equal size. In this way, the memory management overhead could be significantly reduced both in terms of time and space, so that under normal conditions most of the delays now stem from input/output operations, not from memory management.

4. Performance

To evaluate the performance of *memscore* relative to *phrase-extract*, we tested it on two tasks of different sizes. As a medium size task, we trained a phrase table on the German-English portion of the Europarl corpus (Koehn, 2005). The large task uses all parallel data of the Arabic-English constrained data set of the 2009 NIST Machine Translation evaluation campaign. The experiments were performed on the computing cluster of Fondazione Bruno Kessler (FBK-irst), Trento, on Linux computers with 2.5 GHz Intel Xeon CPUs. The cluster setup at FBK also defined the constraints for the practical usability of the scoring tool and for the comparison with *phrase-extract*. The maximum amount of random-access memory on a single machine is 70 GB. The local disks of the computing nodes are relatively small, so that all data must be read from and written to a network-mounted drive, which has a significant negative impact on the performance of both *memscore* and *phrase-extract*. Temporary files created during the initial and intermediate sorting steps of the *phrase-extract* procedure were kept on the local disk. The *memscore* procedure did not require any sorting steps.

The results of the experiments can be found in table 5. On the cluster hardware at FBK, it is possible to train even a NIST system in memory using *memscore*, even though this is clearly pushing it to the limit. The memory usage of *memscore* is approximately proportional to the size of the phrase table, which in turn depends on the corpus size. In this way, the maximum corpus size that can be handled on a machine with a given

amount of memory can be estimated. The memory consumption of the scoring procedure with *phrase-extract*, on the other hand, is clearly not correlated with the corpus size; indeed, in our example runs it was greater for the smaller corpus. The *score* program itself consumes hardly any memory except for storing the lexical translation table. It is likely that the peak values reported in table 5 are due to the GNU *sort* utility which for some reason settled on a different trade-off between using temporary disk files and increased resident memory usage in the two conditions.

The time required to estimate a phrase table is roughly halved by the use of *memscore*. This time is largely dominated by network input/output operations, and the difference roughly reflects the fact that *phrase-extract* scores the two phrase table halves separately, whereas *memscore* can do it in one step. It should also be noted that, as a result of being I/O-dominated, the timing is very sensitive to the overall load on the machines and the network, a factor not controlled in the experiments, so the indications should be taken with a grain of salt. Experience shows that the actual scoring is very fast compared to loading and saving the data, so it is possible to apply iterative scoring methods even for large data sets without incurring a noticeable performance penalty.

To illustrate this effect, we ran another experiment to determine the cost of a single iteration through the complete phrase table excluding the time to load and save the table. We simulated a simple iterative scoring algorithm performing 200 passes through the whole data. In each pass, an operation identical in cost to a relative frequency computation, composed of looking up the marginal count in the phrase information structure and a division, was executed for every phrase pair. The last row in table 5 reports the average time per iteration, which gives an estimate of the marginal cost of an additional pass through the data in an iterative algorithm once the loading and saving times are accounted for.

5. Future work

In its current state, *memscore* can be a useful tool to speed up the training pipeline of an SMT system when computers with large amounts of random-access memory are available. Its extensible design also makes it easy to implement and test new scoring methods. We hope that the public availability of an extensible scoring framework will reduces the work involved in publishing new scoring methods in the form of ready-to-use implementations.

So far, we have been concentrating on implementing the phrase scoring algorithm, relying on the *moses* training scripts to extract phrases from the word-aligned parallel corpus and to estimate the word-to-word translation probabilities used in calculating lexical weights. It should be relatively straightforward, however, to integrate these steps directly into *memscore*. The scoring tool would build its internal representations directly from the parallel corpus. Phrase extraction files and word-to-word dictionaries would be saved to disk only on request. In addition to making *memscore* more

self-contained, this could also lead to a considerable reduction in the total amount of disk space required, on the one hand, and of disk input/output activity, on the other hand. In a networked environment, where data resides on remote disks, loading only the aligned parallel corpus rather than loading and storing large phrase extraction files could speed up the training process even further.

Another sorting step in the training pipeline could be avoided by making *memscore* output the phrase table in the order required by *processPhraseTable*, which creates binary phrase tables to be used with *moses*. Since *memscore* internally stores the phrase pairs in a hash table, which naturally iterates over its elements in a well-defined order, this only requires defining suitable comparison operators for the phrase representation based on numerical identifiers used internally.

Finally, *memscore* could be extended to estimate lexical reordering tables, so that it would cover the complete training of a phrase-based SMT system given the word alignments.

Acknowledgements

This work was supported by the EuroMatrixPlus project (IST-231720), which is funded by the European Commission under the Seventh Framework Programme for Research and Technological Development. I am indebted to Gabriele Musillo and Marcello Federico for their valuable comments on this paper.

Bibliography

Federico, Marcello, Nicola Bertoldi, and Mauro Cettolo. Irstlm: an open source toolkit for handling large scale language models. In *Proceedings of Interspeech*, Brisbane, 2008.

Koehn, Philipp. Europarl: a corpus for statistical machine translation. In *Proceedings of MT Summit X*, pages 79–86, Phuket, Thailand, 2005. AAMT.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 conference of the North American chapter of the Association for Computational Linguistics on Human Language Technology*, pages 48–54, Edmonton, 2003.

Koehn, Philipp et al. Moses: open source toolkit for statistical machine translation. In *Annual meeting of the Association for Computational Linguistics: Demonstration session*, pages 177–180, Prague, 2007.

Ney, Hermann, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.

Ortiz-Martínez, D., I. García-Varea, and F. Casacuberta. Thot: a toolkit to train phrase-based statistical translation models. In *Proceedings of MT Summit X*, pages 141–148, Phuket, Thailand, 2005. AAMT.

Witten, Ian H. and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4): 1085–1094, 1991.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 97–106

ScaleMT: a Free/Open-Source Framework for Building Scalable Machine Translation Web Services

Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz

Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, Spain

Abstract

Machine translation web services usage is growing amazingly mainly because of the translation quality and reliability of the service provided by the Google Ajax Language API. To allow the open-source machine translation projects to compete with Google's one and gain visibility on the internet, we have developed ScaleMT: a free/open-source framework that exposes existing machine translation engines as public web services. This framework is highly scalable as it can run coordinately on many servers, efficiently managing the resources needed by the engines, and its API is compatible with Google's one. ScaleMT is based on previous efforts to build a web service for the Apertium machine translation toolbox, but we have also tested it with Matxin, another free/open-source transfer-based machine translation engine. Additionally, we have compared ScaleMT to an alternative web service implementation for Apertium, obtaining satisfactory results.

1. Introduction

Machine translation (MT) web services are becoming very useful in the web 2.0 era. One of the key features of web 2.0 applications (O'Reilly, 2005) is that they profit from the contributions of users collaborating in the creation of content. However, linguistic barriers make the massive collaboration and understanding of the contents very difficult. Web applications which integrate machine translation services usually attract users speaking different languages and therefore receive more contributions, as can be seen by the increasing number of web applications relying on the Google Ajax Language API[1]. As a result, the MT engine that provides the service receives a

[1] http://code.google.com/apis/ajaxlanguage/

Corresponding author: vmsanchez@dlsi.ua.es
Cite as: Víctor M. Sánchez-Cartagena, Juan Antonio Pérez-Ortiz. ScaleMT: a Free/Open-Source Framework for Building Scalable Machine Translation Web Services. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 97–106. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0011-9.

high amount of useful feedback and its visibility is increased. In the case of an open-source project, being popular would make people feel more interested in it and even join the community of developers.

Unfortunately, open-source MT projects, such as Apertium (Forcada et al., 2009) or Matxin (Alegria et al., 2007), are usually not designed to act as web services: they are not scalable, since they cannot run coordinately on many computers, and spend many CPU cycles loading resources; and they do not have an easy-to-use and internet-friendly API (Application Programming Interface).

With the aim of overcoming these problems, we introduce ScaleMT, a free/open-source framework that exposes existing machine translation engines as public web services, with an API compatible with Google Ajax Language API. Additionally, it allows the MT engines to be deployed on multiple servers in order to achieve high scalability. ScaleMT is based on a previously developed scalable web service architecture for Apertium (Sánchez-Cartagena and Pérez-Ortiz, 2009), that have been generalised to work with different MT engines. The main advantage of ScaleMT is that the architecture of the engines to be exposed does not need to be changed, although it must meet some requirements explained in section 2.

As the API of the web service has already been described previously (Sánchez-Cartagena and Pérez-Ortiz, 2009), this paper will focus on the ScaleMT architecture and how it can run with different MT engines. Firstly, section 2 will explain which kind of translation engines this framework is focused on. Later, in section 3 we will describe the ScaleMT architecture. After that, section 4 explains the steps needed to add a new translation engine to the service. Section 5 contains the description and results of two different experiments: a comparison between the different MT engines that can run with ScaleMT, and a comparison between ScaleMT and another efficient MT web service that can be found in the literature (Minervini, 2009). Finally, the paper ends with some conclusions that can be drawn from the development of the system and from the experiments, and with a list of future tasks.

2. Translation Engines that Can Profit from this Architecture

ScaleMT has been designed to work with MT engines which have these two features:

1. The translation engine is a process that reads the input text from its standard input and writes the translation to its standard output. It starts to translate before reading the full text from the input and dies when the standard input is closed.
2. Every time the process is launched, it needs to perform some start-up operations that require many CPU cycles before it can translate.

The second feature is very common on transfer-based MT systems such as Apertium or Matxin: rules and dictionaries need to be loaded before the engines can use them to translate the input text; therefore, translating many small texts is extremely inefficient. Our framework reuses the processes of the translation engine, translating

many source texts with the same ones, thus avoiding loading rules and dictionaries time after time. The reuse is possible thanks to the first feature, as will be explained with more detail in the next section.

3. System Architecture

In this section we present the architecture of ScaleMT. Our proposal makes the translation engines more efficient by turning them into *daemons* (that is, processes running in the *background* rather than under the interaction of a user). Besides that, it is able to run on multiple servers thanks to an algorithm which decides which daemons should run on each server and a load balancing method that decides which server should process each request. ScaleMT consists of two main Java applications:

ScaleMTSlave runs on a machine with the translation engine installed and manages a set of running translation engine instances (daemons); it performs the requested translations by sending them to the right daemon.

ScaleMTRouter (*request router*) runs on a web server; it processes the translation requests and sends them to the right `ScaleMTSlave` instance.

The different components of the architecture are explained in detail next.

3.1. Daemonising Engines

As we stated before (see section 2), our framework is designed to work with translation engines that spend many CPU cycles performing start-up operations when they are launched. Since the start-up cost is so high, a daemon to reuse translation engine processes and minimise the amount of start-ups must be found. A daemon is a process that is launched once and can perform many translations. Taking advantage from the first feature described in section 2, we have built a daemon by queueing translation requests, sequentially writing source texts from the requests to the standard input of the translation engine process, and not closing it when there are not requests in the queue. This approach has to deal with two issues that are solved in different ways depending on the translation engine (see section 4):

- Separating the different translations: the process behaves as if it was translating a long text but we need the different translations to be easily isolated from the output of the process.
- Making the daemon translate immediately: input/output implementations in many operating systems and programming languages use buffers for efficiency reasons. It can happen that a translated text is stored in a buffer and not returned until the buffer is completely filled.

A daemon can only translate with the language pair for which it has loaded the data.

3.2. Load Balancing

The *request router* sends each translation request to a `ScaleMTSlave` instance running a daemon for the involved language pair. Choosing that server and fairly distributing the work between all the available ones is called *load balancing*.

`ScaleMTRouter` manages one queue for each language pair. When a request arrives, it is put on the queue corresponding to its language pair. For each queue, there is a *dispatcher thread* that consumes requests from it independently from the other queues, and sends them to the most suitable server. Each request in the queue has an associated CPU cost. The dispatcher thread keeps track of the sum of the CPU costs of the requests that have been sent to each server, but have not been completed yet, namely *waiting rate*. Dispatching works as follows:

1. The dispatcher thread checks whether the lowest waiting rate in the set of servers with a daemon for the associated pair is lower than a particular threshold. If this condition is not held, it waits a short time and executes this step again.
2. It takes the first request from the queue, sends it to the server with the lowest waiting rate, and returns to step 1. Server's waiting rate is updated accordingly.

This way, although queues are independent, work is fairly distributed. If a server is processing many requests for a language pair A, requests for language pair B will take a long time to be processed because both need to share CPU. If there is another server that is not processing A requests, it will translate B requests faster and, consequently, receive more B requests as it will be more often the server with less waiting rate.

3.3. Application Placement

Servers usually do not have enough memory to run a daemon for every supported language pair. Consequently, we should run the daemons which receive more translation requests, and adapt the number of daemons and the power of the machines where they run to the amount of work they have to perform. Additionally, load changes throughout time, so the running daemons should change too. To deal with these problems we have developed a *placement algorithm* based on the work by Tang et al. (2007) that is executed periodically and decides which daemons should run on each server.

The algorithm is widely based on the concept of load. The load is an estimation of the CPU power needed to perform translations measured in *translated characters per second*. Each server has a *load capacity*, estimated by translating a set of texts on it, and a *memory capacity*. Each language pair has a *memory requirement*, estimated once by simply running its daemon and measuring the memory it needs, and a *load requirement*, estimated periodically from the requests received by the service. After new load requirements are estimated, the algorithm is executed to decide how many daemons of each language pair should run on each server, following these guidelines:

- Satisfied load must be maximised, since the load capacity of a server is distributed between the language pairs of the daemons running on it.

- The number of daemons to be started or stopped must be minimised.
- The summation of the memory requirements of the daemons running on a server must not be higher than its memory capacity.

3.4. Scaling

The whole system is able to scale by adding new servers running `ScaleMTSlave`. These servers can be added manually, or we can let a *dynamic server manager* decide when to add or remove them. The servers added by the *dynamic server manager* can be machines from a local network (with SSH access enabled) or Amazon EC2 instances.[2] This component decides when the system needs more servers based on the *placement algorithm* output: if the amount of load satisfied by the placement proposed by the algorithm is lower than the total load demand (because the servers do not have enough CPU capacity or enough memory), then new servers are added.

4. Adding a New Translation Engine

With the aim of achieving engine-independence, ScaleMT uses an XML configuration file for defining relevant information about the translation engines without changing a single line of Java code nor recompiling the project.

Firstly, we must specify the language pairs and formats supported by the engine. Then, the commands which run the translation engine need to be defined. Most translation engines are made of a deformatter module, that removes the format information from the input text; a translation core, that translates the text; and a reformatter, that restores the format information. Therefore, a pipeline of programs can be defined. One of the components of the pipeline (the one with the highest start-up time, usually the core) should be chosen to be kept in execution, and the other components allowed to be executed once for every translation, even with different parameters depending on the translation type. The configuration format allows a high flexibility on the communication between the pipeline components.

The most important part of the configuration file contains the solution to the problems defined in section 3.1:

- The strategy to isolate the different translations in the engine flow consists of adding extra sentences containing an unique identifier before and after the text to be translated. Then, their translation and the unique identifier of the text are searched in the output.
- Ensuring that the translation engine returns the translation immediately and it does not remain stored in buffers involves sending some extra text (*padding sentences*) to the daemon after each translation to completely fill the buffers. A *padding sentence* should be included in the configuration file and the system em-

[2] `http://aws.amazon.com/ec2/`

pirically estimates how many units of that sentence has to send after each translation source to completely fill the buffers and get the translation. To avoid overloading the system, *padding* is not sent unless the daemon local queue is empty, which means that it will not receive more requests soon.

Sample configuration files with detailed comments about each section can be found with the source code of ScaleMT[3]. As a proof of concept, we have evaluated our framework with Apertium and Matxin.

4.1. Apertium

In the Apertium pipeline (Forcada et al., 2009), text is first deformatted, and the format information is put into *superblanks*, special blocks between square brackets that are not translated. The output of the deformatter is then translated by the core, and finally the format is restored by the reformatter. We decided to keep in execution only the core, a decision which has two advantages. First, we can use *superblanks* to separate the different translations, and forget to worry about how the separation sentences will be translated. And second, since the deformatter and reformatter are executed for each translation, we can use the same daemon to translate different formats.

With Apertium there is no need of sending *padding* after each source text to be translated as all the modules in the Apertium pipeline have an mechanism called *null flush* that makes them flush their buffers when they receive a null character in their input. Consequently, the configuration XML includes and option to enable the sending a null character after each translation request.

4.2. Matxin

Matxin pipeline architecture (Alegria et al., 2007) is different from the Apertium one. Matxin deformatter has two outputs: the text without format and the format information. The unformatted text is translated by the core and later joint with the format information by the reformatter. As a consequence of that, we have to keep in execution only the core because a pipeline cannot be built with two flows. We have used as separator sentences unknown words since they do not fire any transfer rule that could waste CPU. As Matxin does not support null flush, we use the *padding* mechanism to ensure that the translation engine returns the translations immediately. We have chosen a long sequence of random characters as *padding* because a single unknown word consumes residual CPU time, and a long word fills the buffers faster.

Additionally, two more problems arose during the adaptation process. First, one of the components of the Matxin pipeline is `iconv`, a process that changes text encoding. It does not write the text with the new encoding until it reads the end-of-file (EOF) character. So, we wrote a modified version of `iconv` that processes input file line

[3] `http://apertium.svn.sourceforge.net/svnroot/apertium/trunk/scaleMT/ScaleMTSlave/sampleconfs`

per line. Second, the Matxin process crashes with some inputs making the daemon running on the top of that process to die. To deal with this problem we have created a mechanism which detects if a process has died and launches it again.

5. Experiments and Results

Note that in order to ensure reproducibility, scripts that automatically perform the following experiments are available[4].

5.1. Efficiency Gain for Apertium and Matxin

First, we will estimate the performance improvement in Apertium and Matxin when translating a text with ScaleMT. The main idea under Amdahl's law (Amdahl, 1967) is that the performance improvement obtained by using some *faster mode* of execution in a program is limited by the fraction of the time the *faster mode* can be used. Analysing the time needed to perform a translation with Apertium or Matxin, we can split it in the time needed to load resources (*start-up time*) and the time needed to perform the actual translation (*translation time*). ScaleMT reduces the start-up time because the time needed to start the whole MT engine process is replaced by a little overhead caused by launching the deformatter and the reformatter, and by translating the separation sentences; but translation time is unchanged. Therefore, the fraction of the time the faster mode can be used corresponds to the start-up time.

The plot on the left of figure 1 shows the time needed to translate input texts of different lengths by four different systems: Apertium (Spanish–English), Matxin (Spanish–Basque), ScaleMT running Apertium (Spanish–English) and ScaleMT running Matxin (Spanish–Basque). The time needed to translate a 0-length text is the start-up time. The plot on the right shows the performance gain in Apertium and Matxin. The experiments have been run with an instance of `ScaleMTRouter` and and a single instance of `ScaleMTSlave` running on the same machine.[5] Following Amdahl's law, Apertium's gain is greater than Matxin's because, for the same source text length, the percentage of time spent in the start-up operations is greater too.

5.2. Comparing with Other Scalable Apertium Web Service Implementations

Different approaches to build scalable MT web services have been proposed. For instance, Minervini (2009) designed `Apertium-service`, an efficient Apertium-based web service, by completely changing Apertium architecture. The key idea under his approach is replacing the original pipeline-based architecture with a multithreaded

[4] `http://apertium.svn.sourceforge.net/svnroot/apertium/trunk/scaleMT/ScaleMTRouter/experiments`

[5] An AMD Turion TL-56 with 2 GB of RAM memory

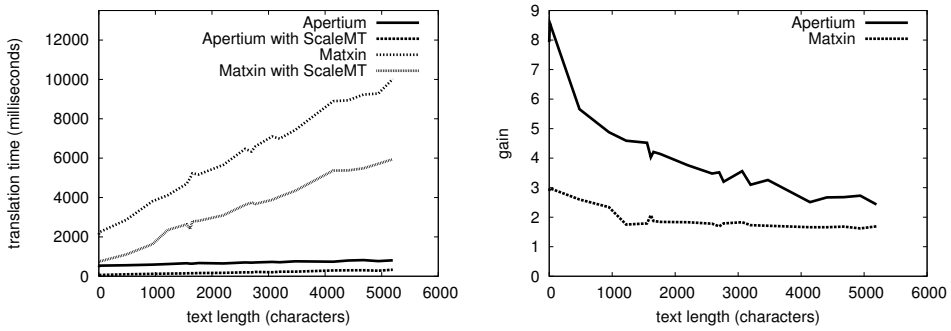

Figure 1. Translation time and efficiency gain with the use of ScaleMT.

resource pool in which linguistic data is kept loaded. We have performed some experiments comparing ScaleMT with `Apertium-service`. They are similar to the experiments originally performed by Minervini (2009) comparing his `Apertium-service` with the predecessor of ScaleMT, but they cover a wider data range.

The plot on the left of figure 2 shows the time needed to translate (English–Spanish) fragments of different lengths of the GPL license text[6] by a single client. The plot on the right shows the average time needed to translate (again from English to Spanish) the preamble of the GPL license text (2531 characters) when requested by different amounts of concurrent clients. Both experiments have been performed on an Amazon EC2 small instance. In the case of ScaleMT, there is only one slave running on the same machine as the router.

5.3. Other Interesting Results

The previous experiments have targeted only a single slave instance. However, it is also important to check if the architecture is able to scale to a high number of slaves. The experiments performed by Sánchez-Cartagena and Pérez-Ortiz (2009) about this topic show that, running on a not very powerful machine (an Amazon EC2 small instance), the router can process up to 19 000 requests per minute. With input texts that do not need a high CPU capacity to be processed (56 characters, Apertium Spanish–Catalan), 20 slaves are needed to perform the work needed by such a high request rate. Since the changes made to generalise the architecture have been mainly focused on the structure of the slaves, that maximum request rate is still valid. However, since translating a text with Matxin needs more CPU capacity, the number of servers needed to manage this rate could be even higher.

[6] `http://www.gnu.org/licenses/gpl.html`

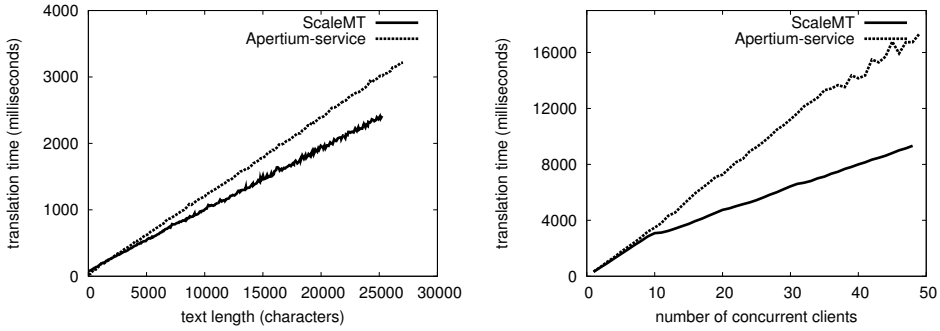

Figure 2. Comparison between ScaleMT and `Apertium-service`.

6. Conclusions

We have developed ScaleMT, a free/open-source framework to automatically create web services from existing MT engines. According to our experiments, ScaleMT is more suitable to work with Apertium than with Matxin. When translating texts of around 500 characters, Apertium performance gain is 5.6, and Matxin's is only 2.6, mainly because the start-up time of Apertium is bigger than the start-up time of Matxin, compared with the time needed to translate a typical text. Additionally, Matxin, as of revision 248, is unstable and unpredictably crashes with some inputs. Consequently, the daemon is started more times than necessary, causing performance loss.

When comparing with `Apertium-service`, the time needed to translate individual texts is similar in both architectures, although ScaleMT performs better with longer texts. This is not a clear advantage because translations requested from web pages to a JSON API are not usually very long. However, when testing the systems in a more realistic scenario (many concurrent clients), ScaleMT outperforms `Apertium-service`. Furthermore, ScaleMT is engine-independent and, consequently, adaptable to future changes in Apertium, while `Apertium-service` is an ad-hoc solution.

ScaleMT is currently under evaluation to be the *official* Apertium web service. Source code for ScaleMT can be downloaded from `http://apertium.svn.sourceforge.net/svnroot/apertium/trunk/scaleMT`.

7. Future Work

It would be interesting to consider ScaleMT to build a web service over the Moses decoder (Koehn et al., 2007). Firstly, it should be checked whether it meets the requirements explained in section 2, and then find the appropriate sentences to separate the

different requests in the pipeline. We originally did not address Moses because there is already a web service implementation for it. However, we could draw interesting conclusions by comparing ScaleMT with the Moses web service.

It is worth improving scalability by increasing the maximum request rate supported by the router. This is not an easy task because it probably would involve having many router instances and synchronising them.

8. Acknowledgements

This work has been partially funded by Google through the Google Summer of Code program and by Spanish Ministerio de Ciencia e Innovación through project TIN2009-14009-C02-01.

Bibliography

Alegria, I., A.D. de Ilarraza, G. Labaka, M. Lersundi, A. Mayor, and K. Sarasola. Transfer-based MT from Spanish into Basque: reusability, standardization and open source. *Lecture Notes in Computer Science*, 4394:374–384, 2007.

Amdahl, G.M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM New York, NY, USA, 1967.

Forcada, M.L., F.M. Tyers, and G. Ramírez-Sánchez. The Apertium machine translation platform: five years on. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 3–10, 2009.

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 *Annual Meeting of the Association for Computational Linguistics: Demonstration Session*, 2007.

Minervini, P. Apertium goes SOA: an efficient and scalable service based on the Apertium rule-based machine translation platform. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 59–66, 2009.

O'Reilly, T. What is web 2.0. In *Design Patterns and Business Models for the Next Generation of Software*, `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`, 2005.

Sánchez-Cartagena, V.M. and J.A. Pérez-Ortiz. An open-source highly scalable web service architecture for the Apertium machine translation engine. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 51–58, 2009.

Tang, C., M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, pages 331–340. ACM, 2007.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 107–116

Integrating Output from Specialized Modules in Machine Translation

Transliterations in Joshua

Ann Irvine[a], Mike Kayser[b], Zhifei Li[a], Wren Thornton[c], Chris Callison-Burch[a]

[a] Center for Language and Speech Processing, Johns Hopkins University
[b] BBN Technologies
[c] Cognitive Science Program, Indiana University

Abstract

In many cases in SMT we want to allow specialized modules to propose translation fragments to the decoder and allow them to compete with translations contained in the phrase table. Transliteration is one module that may produce such specialized output. In this paper, as an example, we build a specialized Urdu transliteration module and integrate its output into an Urdu–English MT system. The module marks-up the test text using an XML format, and the decoder allows alternate translations (transliterations) to compete.

1. Introduction

The phrase tables used in statistical machine translation (SMT) systems are often incomplete, and they may not take full advantage of the linguistic knowledge that we have about a language. However, many data-driven NLP tools exist for specific linguistic tasks. For this reason, it is often useful to create specialized modules that employ other methods of translation not so dependent on the training text. Such modules may include noun phrase taggers and translators (Koehn and Knight, 2004), morphological analyzers, modality taggers and translators,[1] and transliteration systems. These modules may then be integrated into the MT pipeline (Dugast et al., 2007; Yang and Kirchhoff, 2006).

[1] Verbal modality expresses the notions of possibility, necessity, permission, and obligation

Corresponding author: annirvine@gmail.com
Cite as: Ann Irvine, Mike Kayser, Zhifei Li, Wren Thornton, Chris Callison-Burch. Integrating Output from Specialized Modules in Machine Translation: Transliterations in Joshua. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 107–116. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0009-3.

Previous SMT systems have integrated the subtask output of specialized modules using an XML-markup on input text (Koehn, 2004; Senellart et al., 2003). Here we present an XML format to integrate the output of our specialized transliteration module into the Joshua decoder (Li et al., 2009a). It necessarily differs from the XML schemes used in phrase-based decoders because Joshua is a parsing-based decoder. We illustrate its use with an example transliteration module.

2. Decoding Constraints

Joshua (Li et al., 2009a) is an open source[2] SMT system that uses synchronous context free grammars (SCFGs) as its underlying formalism. SCFGs provide a convenient and theoretically grounded way of incorporating linguistic information into statistical models of translation. Joshua implements all the essential algorithms described in (Chiang, 2007) and supports Hiero-style rules (Chiang, 2005) as well as richer syntax augmented rules (Zollmann et al., 2008). The version of Joshua that we have used in this work incorporates the grammar extraction software that comes as part of their open source SAMT toolkit.[3] Joshua translates by applying the extracted SCFG rules to the source language text using a general chart parsing framework (Li et al., 2009b).

A probabilistic SCFG consists of a set of source-language terminal symbols T_S, a set of target-language terminal symbols T_T, a set of nonterminals N that is shared between both languages, and set of production rules of the form

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

where $X \in N, \gamma \in [N \cup T_S]*$ is a (possibly mixed) sequence of nonterminals and source terminals that form the lefthand side of the rule, $\alpha \in [N \cup T_T]*$ is a (again possibly mixed) sequence of nonterminals and target-language terminals that form the right-hand side of the rule, and $\sim$ is a one-to-one correspondence between the nonterminals of γ and α. w is a weight for the production rule.

To support integrating specialized modules we introduced the ability to specify alternate translation rules in the document to be translated. In order to support both the use of alternate translation rules and regular decoding (without alternate rules), we introduced a new parameter in Joshua's configuration file for specifying which parser to use on the input file. Each input file parser reads in the file and emits a sequence of segments to be translated. In order to avoid storing the whole file in memory, the sequence is returned as a co-iterator.[4] By using a co-iterator, the sequence is produced lazily and consumed on-line, invoking the chart parser as a co-routine.

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

[3] http://www.cs.cmu.edu/zollmann/samt/

[4] We use some object instance which implements the `joshua.util.CoIterator` interface. Both iterators and co-iterators are examples of abstractions over enumerations. Whereas an iterator captures the notion of producing the elements, a co-iterator captures the notion of consuming those elements.

To avoid interrupting translation in the middle of a file, we want to detect malformed files before translation begins. So in order to detect malformed file errors eagerly, the file is read once with a co-iterator that consumes the segments but does nothing with them, and then re-read to produce segments for the chart parser.

Each translation segment consists of an ID, a source sentence, and a collection of spans covering the sentence where each span contains a collection of constraints. Spans containing only soft constraints are allowed to overlap, whereas hard constraint spans may not overlap. Constraints are drawn from three types: lefthand side (LHS) constraints, righthand side (RHS) constraints, and rule constraints. LHS constraints are hard constraints specifying that the span be treated as a specified nonterminal, thus filtering the regular grammar to generate translations only from that nonterminal. One use for LHS constraints is to integrate chunking or tagging information before decoding. RHS constraints are hard constraints filtering the regular grammar such that only rules generating the desired translation can be used. A use for RHS constraints would be integrating word sense disambiguation before decoding. Rule constraints specify a new grammar rule including a LHS nonterminal, source RHS (derived from the source sentence), target RHS, and feature values. Rule constraints can be either hard or soft; if they are hard, they override the regular grammar; if they are soft, they are considered an addition to the regular grammar and will compete with regular rules. Rule constraints for any given span must be marked collectively as all hard or all soft.

2.1. XML Markup

The XML format follows straightforwardly from the specification of segments. The XML file must be valid XML, and thus must have a root element. Underneath the root element are some number of `<seg>` tags with a required `id` attribute that specifies the input segment number. The source sentence is given as raw text under the `<seg>` tag. Each `<seg>` tag may contain one or more `<span>` tags with required `start` and `end` attributes and an optional `hard` attribute for rule constraints. Each `<span>` tag must contain one or more `<constraint>` tags each of which contains an optional `<lhs>` tag, containing nonterminal text, followed by an optional `<rhs>` tag with an optional `features` attribute and containing target text. Any other tags are ignored by the XML parser.

This specification of the XML format is overly liberal and could admit files which are non-sensical or which cannot be represented internally. In order to rule out such files, the generated objects are run through a type checker to ensure semantic validity. The type checker verifies the following invariants:

- Each constraint adheres to one of the three types, thus it has
  - only a `<lhs>` tag, or
  - only a `<rhs>` tag with no `features` attribute, or
  - a `<lhs>` tag, a `<rhs>`, and a `features` attribute

- For each span,
  - the start and end indices are within the width of the sentence
  - the start index is smaller than the end index
- There are no overlapping hard spans

While the `features` attribute is considered optional in terms of the DTD for the XML grammar, that is only because DTDs are unable to capture the dependency relation between the three valid constraint types.[5] If both `<lhs>` and `<rhs>` tags are provided but there is no `feature` attribute, this is considered a type error since the constraint does not belong to any of the three types: LHS constraints do not have `<rhs>` tags, RHS constraints do not have `<lhs>` tags, and rule constraints require a `features` attribute.

3. Decoder Integration

To enforce the three kinds of constraints (i.e., rule, LHS, and RHS) during decoding, we modified the regular chart-based decoding algorithm in Joshua. Rule constraints can be hard or soft. A rule constraint provides a new translation option for a source span, in addition to those translation options (hereafter called grammar-based translations) provided by the regular grammars. If the rule constraint is hard, all the grammar-based translations will be disallowed in the final translation output. Otherwise, the new translation option will compete with those grammar-based translations, in a probabilistic manner. Different from a rule constraint, LHS and RHS constraints are always hard, meaning that a grammar-based translation will be disallowed if its LHS or RHS does not match the LHS or RHS constraint.

Figure 1 presents the modified algorithm, where lines 1, 4, and 5 are added to the regular chart-parsing algorithm in order to support manual constraints. As shown in line 1, the algorithm first adds the rule constraints (regardless of being soft or hard) into the chart so that the decoding algorithm will consider these rule translations as candidate translations. To support the **hard** constraints, the algorithm will run through two filtering processes. In line 4, if a span is within the coverage of a **hard** rule constraint, all the grammar-based rules applicable to this span will be disallowed. Similarly, all the applicable grammar rules that do not match any LHS or RHS constraints for the span will be filtered out, as shown in line 5.

4. Transliteration Module

Here we present a specialized transliteration module that uses Joshua's new XML markup. We developed an Urdu-English transliterator, which is useful because our

[5] XML Schema and RelaxNG are also unable to capture all the necessary dependency relations. Even if they can capture the tag and attribute dependencies between the three constraint types, type checking would still be necessary due to the context-sensitive restrictions on valid start and end indices depending on the length in words of the source text.

DecodingWithConstraints(*grammars*, *sentence*, *constraints*)

1 Add the rule constraints into the Chart
2 For each span $[i, j]$ with increasing length (i.e., $j - i + 1$)
3 Identify applicable grammar rules for the span
4 Filter the grammar rules based on **hard** rule constraints
5 Filter the grammar rules based on LHS and RHS constraints
6 Add the surviving grammar rules into the chart

Figure 1. Constraining Chart-based Decoding with Manual Constraints.

LDC Urdu Language Pack bilingual parallel corpus [6] has only 88,108 sentence pairs with 1,586,065 English tokens and 1,664,409 Urdu tokens. In this data, we observed that 2% of words in a development set were out-of-vocabulary (OOV) with respect to the training bitext. With the help of a human annotator, we found that approximately 33% of these words were phonetically transliterable; for example, proper names or borrowed words. Introducing a module for generating transliterations and integrating that output into the output of an end-to-end MT system clearly has the potential to improve performance.

4.1. Basic Framework

We treat transliteration as a *monotone character translation* task, similar to the work of (Knight and Graehl, 1997). We used the Joshua MT system to build an Urdu transliteration module using a semantically-informed framework, described below, and several sources of transliteration pairs for training.

At training time, given a list of Urdu–English name pairs, we first perform character-to-character alignment using the freely available Berkeley Word Aligner.[7] Next, we find character-sequence pairs which conform to the alignment graph for a word pair; these are analogous to phrase pairs in phrase-based statistical MT. We build a table of such character-sequence mappings, annotated with translation probabilities. Finally we extract a frequency-annotated list of 1.3 million names from the English Gigaword corpus using a named entity tagger (Finkel and Manning, 2009). We then use this to train a character language model prior. Having trained these components, we use them in conjunction with the off-the-shelf Joshua MT decoder.

During decoding, a novel Urdu word is segmented into sequences of characters, and each character sequence is translated to an English character sequence. Unlike in the closely analogous process of phrasal machine translation, in phonetic transliteration the translated character sequences are never reordered. Transliteration hy-

[6] LDC catalog number LDC2009E12, "NIST Open MT08 Urdu Resources"

[7] http://code.google.com/p/berkeleyaligner/

potheses are scored using a log-linear model which makes use of character sequence translation scores and a character-LM prior score. The result is a single English phonetic gloss of the Urdu word.

4.2. Semantically-targeted transliteration

We trained two transliteration systems, one for person names and one for all other semantic types (including non-names). These two systems shared all components except for the character LM and the dataset used for decoder weight tuning. For the person-name transliteration system, we trained our character language model from the large list of English person names automatically extracted from the Gigaword corpus. In the non-person-name transliteration system, we trained a model from a large list of English words, without regard to semantic type.

4.3. Training the module

In order to train the transliteration module, we gathered pairs of names that were likely to be transliterations of one another. We obtained unique name pairs from three sources: the Urdu–English parallel corpus (about 2,000 pairs, extracted from alignments and checked by a human annotator), Amazon's Mechanical Turk system (over 12,000 pairs), and linked people pages from the Urdu and English Wikipedias (about 1,000 pairs). Our final system used about 15,000 pairs of Urdu–English transliterated name pairs.[8] Transliteration performance improved with increasing amounts of training data, and our final module outperformed the baseline system available from the LDC Urdu Language Pack.

4.4. Integrating the module

When using the rule constraint mechanism to add externally constructed theories to Joshua's search space, it is necessary to specify a left-hand-side nonterminal for each rule. Since this nonterminal label is used by Joshua in decoding, it is helpful to choose a label which accurately represents the grammatical content of the covered phrase. Rather than compute a single guess for this label, we add multiple arcs to the search space, each with a different nonterminal. We use the automatically determined named-entity-category of the source word to construct the set of possible labels for a transliterated word.

In particular, before decoding we compute from training the N most frequent target nonterminal labels assigned to low-frequency words of each source name category (the name categories are: Person, Location, Organization, Geopolitical Entity, Facility). At decoding time, we use an automatic name tagger to compute the semantic category of each source word. For every transliteration candidate, we add N arcs to

[8] The complete name pair list is freely available at http://www.clsp.jhu.edu/~anni/

```
<seg id="20">
پروفیسر الزیت گارڈنر کے مطابق ڈائبٹنگ کرنے والے افراد قلو سے ن صرف بیٹ جلد متاثر وجائیں گے بلک ان کو صحتیاب ونے میں بھی زیاد وقت لگے گا .
<span start="2" end="3" hard="false">
 <constraint>
   <lhs>[NNP-PERSON]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0.3361808473;
   -9.0926338104;-0.5569314994;0">gardner</rhs></constraint>
 <constraint>
   <lhs>[NNP-PERSON]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0;
   -10.4980070804;-0.7064663926;0">gardiner</rhs></constraint>
 <constraint>
   <lhs>[NNP-PERSON]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0;
   -11.1132723471;-0.8203325663;0">gardener</rhs></constraint>
 <constraint>
   <lhs>[NNP-PERSON]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;0;0
   ;0;0">##UNKNOWN##</rhs></constraint>
 <constraint>
 <lhs>[NN-PERSON]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0.3361808473;
   -9.0926338104;-0.5569314994;0">gardner</rhs></constraint>
 . . . . .
</span>
<span start="5" end="6" hard="false">
 <constraint>
   <lhs>[NNP]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0.003016175;
   -13.1667265966;-0.863364734;1">datong</rhs></constraint>
 <constraint>
   <lhs>[NNP]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0;-12.3903889565;
   -0.7339666773;1">dating</rhs></constraint>
 <constraint><lhs>[NNP]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;0;
   0;0;0">##UNKNOWN##</rhs></constraint>
 <constraint>
   <lhs>[NN]</lhs><rhs features="0;0;0;0;0;0;0;0;0;0;0;0;0;1;0;0;-6.907755279;0.003016175;-13.1667265966;
   -0.863364734;1">datong</rhs></constraint>
 . . . . .
</seg>
```

Figure 2. An example of XML markup on Urdu text. Each span has 21 associated feature weights. The first 13 features represent the feature space of traditionally extracted Joshua translation rules. Since transliteration rules do not exist in the same feature space, these feature values are always set to 0 for transliteration rules.

the search space, each with a different label from the semantically-targeted set. In our experiments, we use the value N=5.

An example of the XML markup for an input Urdu sentence is shown in Figure 2. In this example, two word spans are tagged with constraints and hypothesis transliterations. Each span and transliteration hypothesis is tagged with the N most frequent target nonterminal labels (e.g. NNP, NN, etc. for a PERSON tag).

5. Results

We tested the impact of transliterator integration in a small number of blind submissions to the NIST MT09 Urdu–English evaluation. We integrated the transliterator into the current Joshua system. In one experiment, we transliterated all low-frequency words, while in a second experiment we transliterated only low-frequency

Joshua Translation System	NIST MT09 BLEU Score
No Transliteration	.2958
Transliterate names only	.2980
Transliterate all types	.3010

Table 1. Impact of transliteration on BLEU in submissions to NIST MT09 evaluation.

person names. Our baseline for comparison was the identical Joshua system without transliterations.

We compared the baseline against the transliteration-aware systems both quantitatively and qualitatively. In a quantitative comparison, whose results are in Table 1, transliteration yielded a small but notable BLEU improvement. As shown in the table, transliterating words of all semantic types yielded slightly better performance than transliterating only words marked as person names.

We also qualitatively compared the best transliteration-aware system with the baseline system via manual inspection of decoder output. As expected, some sentences showed clear improvement via the increased lexical coverage allowed by the transliteration model, while other sentences showed little benefit. In some sentences, the transliteration model hypothesized incorrect transliterations for OOV words. More effectively filtering such incorrect translation options, such as through a more developed measure of confidence, is a potential avenue for future work. Tables 2 and 3 show examples of Joshua decoder output with and without the transliteration feature.

6. Conclusion

In this work, we created an XML format to markup the output of specialized subtask modules and integrate alternate translations into the SMT decoder. We created a transliteration module using a character-based statistical MT system and several thousand pairs of transliterated words. The results are promising. In particular, a qualitative analysis suggests that the transliterations were able to appropriately compete with the phrase-based translation output. This work has also opened the door to integrating additional specialized translation modules. Such modules have a potential to increase translation performance, particularly in low-resource conditions.[9]

[9]This research was supported in part by the European Commission through the EuroMatrixPlus project, by the US National Science Foundation under grant IIS-0713448, and by the Human Language Technology Center of Excellence. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect the views of the sponsor.

Without Transliteration	**[UNKNOWN]** Members said that the economic plan, expensive, and will not be effective.
With Transliteration	**Republican** members said that the economic plan, expensive, and will not be effective.
Reference	The republican members said that this economic plan is very "expensive" and will not be effective.
Without Transliteration	However, **[UNKNOWN]** said that he **[UNKNOWN]** president to respect their age and are also due to yell at them, but they were saying truth from miles away.
With Transliteration	"However, **Erdogan** said that he respects the **Israeli** President and his age as a result of which they yell at them , but they were saying the truth from miles away.
Reference	However, later **Erdogan** said that he respects **Israeli** President and his age as well which is why he did not yell at him but whatever he was saying was miles away from truth.

Table 2. Examples of improvements from transliteration.

Without Transliteration	In southern germany **[UNKNOWN]** a resident of the area of **[UNKNOWN] [UNKNOWN]** has left behind a big business group
With Transliteration	A resident of the area of **Cuba** in south Germany **Adolf Merkel** has left behind a big business group
Reference	**Adolf Merckle** of southern Germany's **Swabia** area has left a large business group behind

Table 3. Impact of transliteration. Note that the location name "Swabia" was incorrectly transliterated to "Cuba." This example indicates the future room for improvement.

Bibliography

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)*, 2005.

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

Dugast, Loïc, Jean Senellart, and Philipp Koehn. Statistical post-editing on systran's rule-based translation system. In *Proceedings of the Workshop on Statistical Machine Translation, part of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, 2007.

Finkel, Jenny Rose and Christopher D. Manning. Nested named entity recognition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP-2009)*, 2009.

Knight, Kevin and Jonathan Graehl. Machine transliteration. In *Proceedings of the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL-1997)*, 1997.

Koehn, Philipp. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the 6th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2004)*, 2004.

Koehn, Philipp and Kevin Knight. Feature-rich statistical translation of noun phrases. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-2004)*, 2004.

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 Workshop on Statistical Machine Translation, part of the Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL-2009)*, 2009a.

Li, Zhifei, Chris Callison-Burch, Sanjeev Khudanpur, and Wren Thornton. Decoding in joshua: Open source, parsing-based machine translation. In *Prague Bulletin of Mathematical Linguistics*, number 91, 2009b.

Senellart, Jean, Christian Boitet, and Laurent Romary. XML machine translation. In *Proceedings of the 9th Machine Translation Summit*, 2003.

Yang, Mei and Katrin Kirchhoff. Phrase-based backoff models for machine translation of highly inflected languages. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, 2006.

Zollmann, Andreas, Ashish Venugopal, Franz Och, and Joy Ponte. A systematic comparison of phrase-based, hierarchical and syntax-augmented statistical MT. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING-08)*, 2008.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 117–126

The Machine Translation Toolpack for LoonyBin: Automated Management of Experimental Machine Translation HyperWorkflows

Jonathan H. Clark[a], Jonathan Weese[b], Byung Gyu Ahn[b], Andreas Zollmann[a], Qin Gao[a], Kenneth Heafield[a], Alon Lavie[a]

[a] Language Technologies Institute, Carnegie Mellon University
[b] Center for Language and Speech Processing, Johns Hopkins University

Abstract

Construction of machine translation systems has evolved into a multi-stage workflow involving many complicated dependencies. Many decoder distributions have addressed this by including monolithic training scripts – `train-factored-model.pl` for Moses and `mr_runmer.pl` for SAMT. However, such scripts can be tricky to modify for novel experiments and typically have limited support for the variety of job schedulers found on academic and commercial computer clusters. Further complicating these systems are hyperparameters, which often cannot be directly optimized by conventional methods requiring users to determine which combination of values is best via trial and error. The recently-released LoonyBin open-source workflow management tool addresses these issues by providing: 1) a visual interface for the user to create and modify workflows; 2) a well-defined logging mechanism; 3) a script generator that compiles visual workflows into shell scripts, and 4) the concept of Hyperworkflows, which intuitively and succinctly encodes small experimental variations within a larger workflow. In this paper, we describe the Machine Translation Toolpack for LoonyBin, which exposes state-of-the-art machine translation tools as drag-and-drop components within LoonyBin.

1. LoonyBin Background

Empirical research in machine translation has become a complex multi-stage process with many stages being run under multiple experimental conditions (i.e. with different corpora and different sets of hyperparameters). The management of such

Corresponding author: jhclark@cs.cmu.edu
Cite as: Jonathan H. Clark, Jonathan Weese, Byung Gyu Ahn, Andreas Zollmann, Qin Gao, Kenneth Heafield, Alon Lavie. The Machine Translation Toolpack for LoonyBin: Automated Management of Experimental Machine Translation HyperWorkflows. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 117–126. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0002-x.

workflows presents a real challenge in terms of keeping results organized, analyzing results at every stage, and automating the workflow.

For example, in syntactic statistical machine translation, a typical experiment consists of over 20 tools with a complex network of dependencies spanning multiple machines or even clusters of machines. Parsing and phrase extraction might be run on a large cluster of hundreds of low-memory machines, preprocessing and word alignment might be run on a local server, while tuning and decoding might be done on a small cluster of large-memory machines. Further, this system might be run for two language pairs and using 10 sets of features in the translation model to verify some experimental hypothesis.

With these needs in mind, LoonyBin (Clark and Lavie, 2010) accommodates workflows that:

- span various machines, clusters, and schedulers
- involve many separate tools, which can be invoked by arbitrary UNIX commands
- have components that are run multiple times under multiple conditions
- evolve quickly with tools frequently being added, removed, and swapped

LoonyBin accomplishes this by providing the following advantages over current common practices:

- associating sanity checks and logging directly with tools, separating these from ad hoc wrappers and automation scripts
- maintaining a cleanly organized directory structure for each step and each condition under which a step is run
- providing a resume-on-failure mechanism for every stage in the pipeline
- making it easy for those without a detailed knowledge of each tool's internals to run the system by providing textual descriptions of each parameter, input file, and output file in a graphical workflow designer
- automatically copying required files between machines/clusters via SSH
- compiling workflows into shell scripts, a medium already in widespread use by NLP researchers

1.1. Workflow Semantics

We now discuss the representation of workflows in LoonyBin. In their most basic form, LoonyBin represents workflows as Directed Acyclic Graphs (DAGs). In this form, each vertex represents a TOOL, which produces output files given input files and parameters, and directed edges indicate relative temporal ordering of tools and information flow (files or parameters) by mapping the output of one tool to the inputs of the next. A TOOL DESCRIPTOR defines the commands necessary to run a tool given inputs, outputs, and parameters. Custom tool descriptors can be implemented via simple user-defined Python scripts that generate shell commands. These tool descriptors contain PRE-ANALYZERS to check the sanity of the inputs and log information

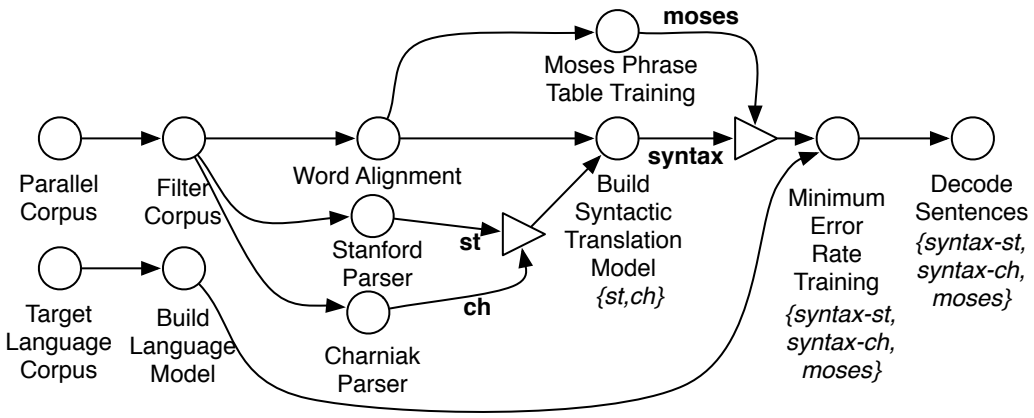

Figure 1. A simplified version of the CMU StatXfer system HyperWorkflow for the GALE Phase 4 Machine Translation Evaluation showing the multiple experiments that were run

and POST-ANALYZERS to check the sanity of the output files, log information about the outputs, and extract log data from any third-party log file formats.

1.2. HyperWorkflow Semantics

LoonyBin also represents the running of workflows under multiple experimental conditions (i.e. with different input files or parameters). We call this a HYPERWORKFLOW. A HyperWorkflow contains REALIZATION VARIABLES, which introduce variations into a shared workflow. Each realization variable can take on a REALIZATION VALUE, which is a set of files and parameters. For instance the realization variable "language model file and order" could take on the realization value {english.txt, 4}. Finally, a REALIZATION INSTANCE is a regular workflow unpacked from a hyperworkflow; it is a configuration of a hyperworkflow such that all realization variables have been assigned a particular realization value. Hyperworkflows are useful for performing exploration of hyperparameters, ablation studies, variation of input corpora, etc.

For HyperWorkflows, we use a HYPERDAG, the hypergraph formulation of a DAG. shown in Figure 1. In LoonyBin, a HYPEREDGE is an edge originating from a PACKING NODE (displayed as a triangle in Figure 1) , which is used to introduce a realization variable. These packing nodes act like a switch to select one of its input edges so that each edge feeding a packing node can create a new realization variable in the workflow. These realization variables are then propagated through the remainder of the workflow. Where multiple realization variables meet, LoonyBin produces the cross-product of their realization values. A HyperDAG is a packed representation of

multiple workflow DAGs and a realization instance is a particular unpacked instance of a workflow. For instance, in Figure 1 edges *st* and *ch* enter a packing node and then propagate realization values *st* and *ch*. By representing workflows in this way, we avoid rerunning steps having the same experimental conditions.

1.3. Standardized Logging and Organized Directory Structure

While being able to automatically execute and reproduce workflows is good, simply completing the job is not enough. We also want to know where the output files came from and some aggregate facts about them. LoonyBin provides a framework for automatically calculating such information and storing it in a uniform format: tab-delimited key-value pairs form a single record, and each record is newline-delimited, making it easy to process these log files using standard command-line tools or scripts. Finally, the log files for all antecedent steps of the same realization instance are concatenated together so that all information from all steps run under a single experimental condition is collected in one place.

Since the user might want to run further analysis later, it is important to be able to easily find the data itself. To accommodate this, LoonyBin maintains a highly organized directory structure for each workflow. Under a master directory, LoonyBin creates a directory with the name of each vertex in the hyperworkflow with subdirectories for each realization. If steps were run on remote machines, pointers to those machines and the relevant output files are stored on a central machine.

1.4. Designing and Deploying a Workflow

LoonyBin provides a graphical tool, which lists all tools in browsable tree. Tools can simply be dragged and dropped into the workflow as vertices and edges can be drawn by dragging arrows between these vertices.

Once a workflow has been designed, LoonyBin can then compile it into an executable shell script. Thus, the only requirement on the machine that executes the workflow is Bash. Before any tools are ever executed, the generated script checks that all input files and all directories containing required tools exist. Because LoonyBin handles all filenames other than the initial inputs, this eliminates the common issue of pipelines crashing due to typos in file and directory names. The generated script will log into remote machines, copying files and executing processes as necessary.

2. A Machine Translation Toolpack

While LoonyBin provides a mechanism for combining tools into workflows, it does not in itself enable the use of tools. For this, we need tool descriptors, which give LoonyBin 1) what inputs, outputs, and parameters a tool requires 2) analyzers that extract aggregate information from output files and perform sanity checks and 3) documentation on the tool that is shown to the user in the graphic interface. The primary

purpose of the MT Toolpack is to provide these descriptors, their analyzers, and common workflows that put the tools together.

2.1. Installation and Configuration

First, we will set up the DESIGN MACHINE where the visual workflow designer will be used to compile workflows into scripts (e.g. a personal laptop). The only dependency on this machine is Java since the Python tool descriptors are executed via Jython. On this machine, download the latest version of LoonyBin and the MT toolpack[1] and extract the tarballs in the same location. You should now have a `LoonyBin` directory that contains a `tool-packs` directory.

Next, we will set up the EXECUTION MACHINES where the compiled workflow script will be run (e.g. head nodes of various clusters). There, download the MT toolpack and extract the tarball, but also execute the installer script `install-dependencies.py`. This will install *only* the tool binaries, not their dependencies. Other dependencies that must already be installed on the machine include: Python (for the installer), Perl (various), Ruby (Multi-Metric Scorer, MEMT), Java (various), Hadoop (SAMT and Chaksi), Boost (MEMT), and Boost Jam (MEMT). The installer will install these binaries in the user-specified directory and also create a PATHS DIRECTORY, which tells LoonyBin where to find the tool binaries on each execution machine. You can prevent a given tool *X* from being installed by using the `--without-X` switch.

LoonyBin can be launched on most platforms by double-clicking the LoonyBin.jar file. Alternatively, it can be invoked with `java -jar LoonyBin.jar`.

2.2. Creating a Workflow

In this section, we describe the creation of an example workflow. This is done on the DESIGN MACHINE, which need not have any network connection to the machines on which the workflow will run. In "editing" mouse mode, select the "manual filesystem" tool from the panel on the left and then click in the center window to create a vertex in the workflow. Use the panel on the right to give the vertex the name `100-files` (the number in the name is just to help us remember what order the steps were run when looking at the names of vertex subdirectories on the file system) and set the `fileNames` parameter to `example1.txt`. Next, add the Head tool from the left toolbox into the workflow and name it `200-take-head`. Create an edge between the vertices by dragging and, in the Add Edge Dialog that appears, connect `example1.txt` to `corpusIn`.

While we could generate a working script from the workflow created so far, we will continue on and create a HyperWorkflow that demonstrates how to "experiment" with the effect head on 2 different files.

Right-click on the edge from `100-files` to `200-take-head` and select `remove vertex`. Next, add another manual filesystem vertex just as above except with the file-

[1] LoonyBin and the MT Toolpack are available at http://www.cs.cmu.edu/~jhclark/loonybin/

names as `example2.txt` and call it `110-different-files`. Create an OR vertex using the OR tool and give the vertex a unique name. Create a hyperedge from `100-files` to the OR vertex by dragging and, in the Add Edge Dialog that appears, connect `example1.txt` to `OR` and press OK. Similarly, connect `110-different-files` to the OR vertex, and in the dialog connect `example2.txt` to `example1.txt` to indicate that these 2 files will be fulfilling the same role in subsequent steps. Now, in "selecting" mouse mode, click on each of the hyperedges and, using the right panel, name them `one` and `two`, respectively. Finally, draw an edge between the OR vertex and `200-take-head` and connect `example1.txt` as the input of `corpusIn`. You will notice that all of the realization names now appear under the new tool vertex. The tool will be run once for each realization using the inputs from each realization edge.

If you wish multiple tools to feed into the same realization variable, you can give the same name to multiple hyperedges feeding into a single packing vertex. Much like each realization instance had different input files above, you can conduct parameter sweeps using multiple Parameter Boxes from the tool tree on the left; each of the parameter boxes can specify a different set of parameter values to be passed to a tool.

2.3. Generating and Running Workflow Script

LoonyBin allows you to design your pipeline on one machine (the DESIGN MACHINE) and then execute the generated bash script on another machine such as a server – hereafter the HOME MACHINE. The home machine will use passwordless SSH to contact any other remote EXECUTION MACHINES (see Section 2.1).

The "Generate bash script" dialog will ask you for this path of the LoonyBin scripts on the home machine. Also, you need to tell LoonyBin a base directory on the home machine where log data and pointers to output data generated during workflow execution will be placed (see Section 1.3). You should also specify the path and name of the bash script that will be generated. We recommend a `.work` extension. Finally, you can give LoonyBin a space-separated list of email addresses to notify when the pipeline either fails or succeeds. Now just copy the bash generated bash script to the home machine you specified and execute it by passing the `-run` flag. All required input files for each step will automatically be transferred to the proper machine before the tool is executed.

3. Included Tools

We now turn to describing the tools that are included in this MT Toolpack. Since LoonyBin provides documentation within the visual workflow designer for each parameter and file of each tool, we will not focus on the low-level details of the tools here. Instead, we discuss the high-level models they implement and what design decisions were made to incorporate each tool into LoonyBin. In general, the style of LoonyBin is to split tasks into as many LoonyBin tools. This allows easy embedding of novel tools,

resumption on failure, analysis of intermediate results, and sharing partial results in a dynamic programming fashion when later models are run with different parameters.

3.1. MGIZA and Chaksi

MGIZA is a multi-threaded word alignment tool based on GIZA++ (Och and Ney, 2003) that utilizes multiple threads to speed up the time-consuming word alignment process. It also supports forced alignment (the process of aligning an unseen test set given trained models) and incremental training with existing models. It can be distributed over a cluster via its integration with Chaksi, a Hadoop MapReduce training framework for phrase-based machine translation. In addition to word alignment, Chaski supports training of Moses-compatible phrase tables and lexicalized reordering models. In our experience, Chaksi has reduced the time to produce a translation model from parallel data from 4 to 5 days to 9-10 hours. For the initial release of LoonyBin we include tools for generating word classes, both Chaksi and MGIZA versions of the most used word alignment models 1/HMM/3/4, and a phrase table builder. Each of these alignment models is exposed as a separate tool to provide the benefits described above in Section 3.

In building LoonyBin MT tools, we aim to encourage best practice. For instance, MGIZA uses the expectation maximization (EM) algorithm to train word alignment models. In every iteration, the sentences are first aligned using the model parameters from previous step, and then the posteriors are collected and re-normalized to generate models for next step. Therefore, the final alignment output is aligned using the model from second-to-last step instead of the final model. Thus, neither concatenating the sets nor force-aligning using the final model is a good comparison for the way the final model was actually aligned. To encourage proper evaluation of word alignments (by using the second-to-last set of EM parameters), we clearly label the output files that should be used for forced alignment in each tool.

3.2. Berkeley Aligner

The Berkeley Aligner provides an implementation for joint or independent training of IBM Model 1, the HMM alignment model, a syntactic variant of HMM, and a novel symmetrization technique called competitive thresholding (DeNero and Klein, 2007). The aligner provides a supervised inverse transduction grammar (ITG) alignment model (Haghighi et al., 2009). While LoonyBin aims to expose subcomponents as much as possible so that it is easier to combine tools in novel ways, the initial release of the MT toolpack contains only 2 tools for the Berkeley aligner corresponding to the supervised and unsupervised models. In the future, we may attempt to expose each direction, model, and symmetrization heuristic employed in the unsupervised model.

3.3. Joshua

Joshua (Li et al., 2009) is an open-source MT toolkit for synchronous context-free grammar models such as Chiang (2005). It includes suffix array extraction of these grammars from an aligned parallel corpus. The toolkit also includes a built-in subsampler for training on large corpora and an implementation of minimum error-rate training. Each step in the training pipeline is exposed as a separate tool in the LoonyBin MT Toolpack.

3.4. Syntax-Augmented Machine Translation (SAMT)

The SAMT model (Zollmann and Venugopal, 2006) is a synchronous context-free grammar based approach to translation that extends the hierarchical phrase based MT model of (Chiang, 2005) to learn grammars with multiple nonterminals. Grammar rules are extracted from a training sentence pair based on a lattice of its contained eligible phrase pairs and a phrase-structure parse tree of the target sentence, yielding rules such as

NP+SBAR $\rightarrow$ NP , die meine NN zuletzt VBD | NP who last VBD my NN

for a German-to-English translation task, expressing the reordering of the verb triggered by a relative clause. The current release of SAMT uses the open-source Hadoop MapReduce framework to distribute its expensive computations (Venugopal and Zollmann, 2009). Each step in the SAMT training and evaluation pipeline has been wrapped as a separate tool in the LoonyBin MT Toolpack.

3.5. Moses

We replace the `train-phrase-model.perl` from Moses (Koehn et al., 2007) with tools that encapsulates each step such as "build lexical translation table," "construct lexicalized reordering model," and "Run Minimum Error Rate Training" rather than wrapping the entire pipeline. Steps that use GIZA++ are not included in the MT Toolpack since with the release of MGIZA++ and Chaksi, there is little motivation to use GIZA++. For the initial release of the MT toolpack, we do not support factored models.

3.6. Common Evaluation Metrics

We provide a tool that runs some of the most common translation metrics in parallel while transparently handling formatting issues: BLEU (Papineni et al., 2001) as implemented by `mteval-13a.pl` (Peterson et al., 2009), NIST (Doddington, 2002), TER 0.7.25 (Snover et al., 2006), Meteor 1.0 (Banerjee and Lavie, 2005), unigram precision and recall, and length ratio. It accepts a simple input format: flat files with one line per segment, or consecutive lines for multiple references. Aside from translation metrics,

we also include alignment error rate (AER) (Och and Ney, 2003), despite its imperfect correlation with translation quality. In addition to providing the files generated by each metric as output, the LoonyBin tool descriptor places all of these scores in the LoonyBin log giving the benefit of standard formatting.

3.7. Multi-Engine Machine Translation (MEMT)

Multi-engine machine translation (Heafield et al., 2009) combines one-best outputs from different translation systems. Translations are aligned using METEOR (Banerjee and Lavie, 2005) and navigated using these alignments. System-specific weights are learned via tuning with MERT; a separate tuning set works best. Typical gains range from one to five BLEU points above the best system, depending on system diversity and score distribution. MEMT is presented as three tools in LoonyBin: The Meteor aligner, MEMT Tuning, and MEMT Decoding.

3.8. Additional NLP Tools

Since modern MT systems often depend on more basic NLP tools, we have also included a few of these tools in the MT Toolpack. For creating language models, we include SRILM and for creating parse trees, we include the Stanford English parser.

4. Recommendations During Tool Development

LoonyBin aims to make it easy to reproduce results. Well-behaved tool descriptors should write the software version to the log files so that the user knows not only what files were used as input and what tools processed that data, but also what version of the tools were used.

However, research often involves iteratively coding and experimentation. For this, we recommend creating a custom tool descriptor that checks out your branch of a source code management system (e.g. subversion), logs the revision number, compiles the code, and then runs the tool. By doing this, researchers can ensure that results are reproducible[2]. Step-by-step instructions on how to create tool descriptors are included as part of LoonyBin's documentation, but are beyond the scope of this paper.

5. Conclusion

We have presented an open-source Machine Translation Toolpack for LoonyBin. We hope that by releasing this tool pack more research effort may be placed on modeling rather engineering, automation, and logging. Further, we hope that this toolpack encourages future research to include the multiple baseline systems and enables more systematic comparisons between them.

[2] As a side benefit, this encourages the best practice of "commit early, commit often"

Bibliography

Banerjee, S. and A. Lavie. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005.

Chiang, David. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005.

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

DeNero, J. and D. Klein. Tailoring word alignments to syntactic machine translation. In *Association for Computational Linguistics (ACL)*, volume 45, page 17, 2007.

Doddington, G. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, page 145. Morgan Kaufmann Publishers Inc., 2002.

Haghighi, A., J. Blitzer, J. DeNero, and D. Klein. Better word alignments with supervised ITG models. In *Meeting of the Association for Computational Linguistics*, 2009.

Heafield, Kenneth, Greg Hanneman, and Alon Lavie. Machine translation system combination with flexible word ordering. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 56–60, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0x08`.

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. Moses: Open source toolkit for statistical machine translation. In *Association for Computational Linguistics (ACL)*, 2007.

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 *Workshop on Statistical Machine Translation (WMT09)*, 2009.

Och, Franz Josef and Hermann Ney. A systematic comparison of various statistical alignment models. In *Computational Linguistics*, 2003.

Papineni, K., S. Roukos, T. Ward, and W. J Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*, 2001.

Peterson, Kay, Mark Przybocki, and Sébastien Bronsart. NIST 2009 open machine translation evaluation (MT09) official release of results, 2009. http://www.itl.nist.gov/iad/mig/tests/mt/2009/.

Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. A study of translation edit rate with targeted human annotation. In *Proc. of AMTA*, page 223–231, 2006.

Venugopal, Ashish and Andreas Zollmann. Grammar based statistical MT on hadoop. *The Prague Bulletin of Mathematical Linguistics*, 91, 2009.

Zollmann, Andreas and Ashish Venugopal. Syntax augmented machine translation via chart parsing. In *Workshop on Machine Translation (WMT) at ACL*, 2006.

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.

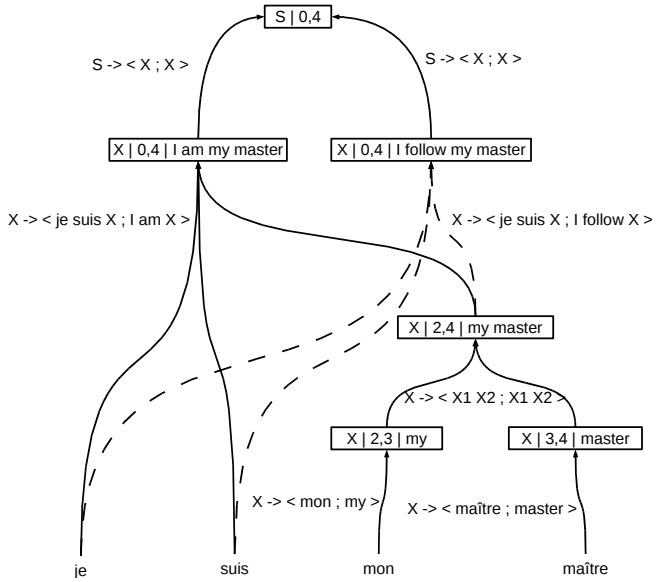

Figure 1. A hypergraph showing two candidate translations of Je suis mon maître.

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/`

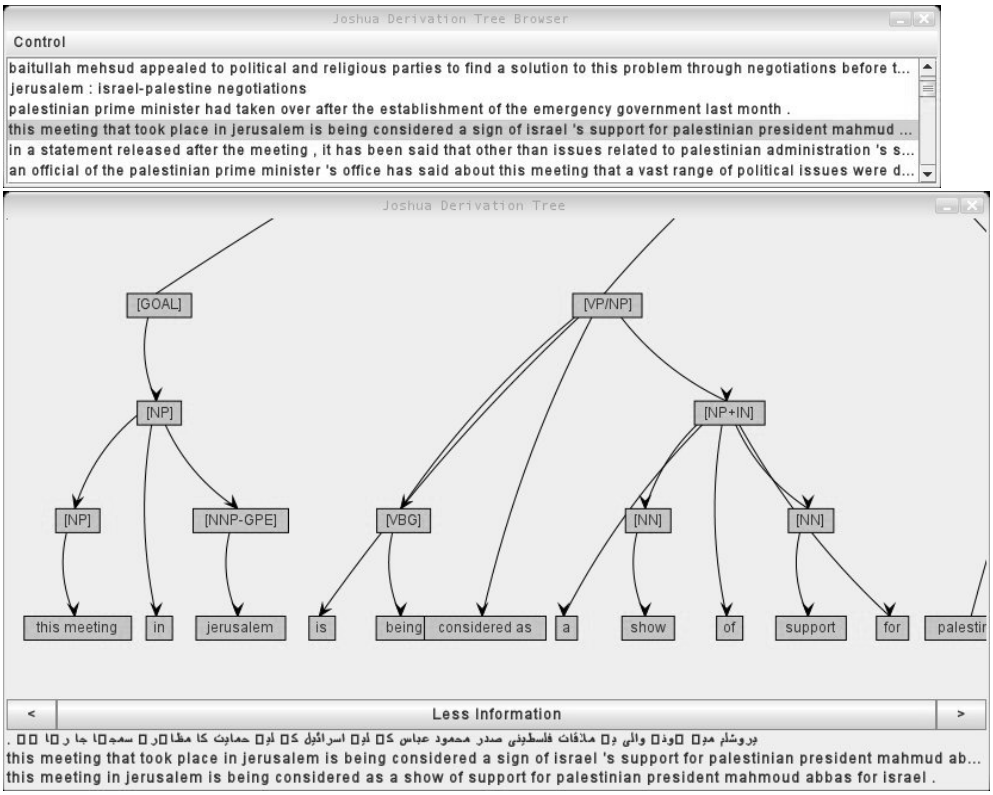

Figure 2. The Derivation Tree browser's sentence selection and tree-viewing windows.

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,

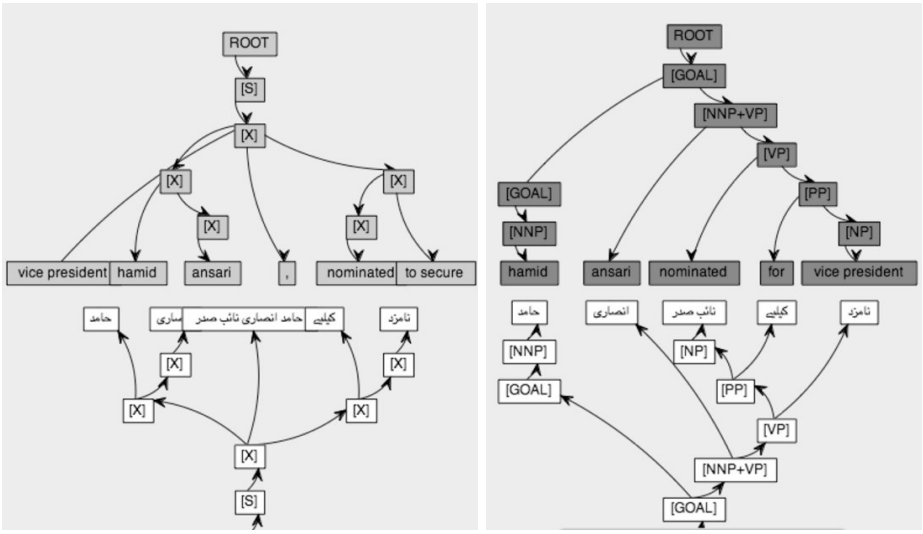

Figure 3. An example visualization of two derivation trees for SCFGs that use a Hiero-style grammar and a syntactically-motivated grammar.

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.

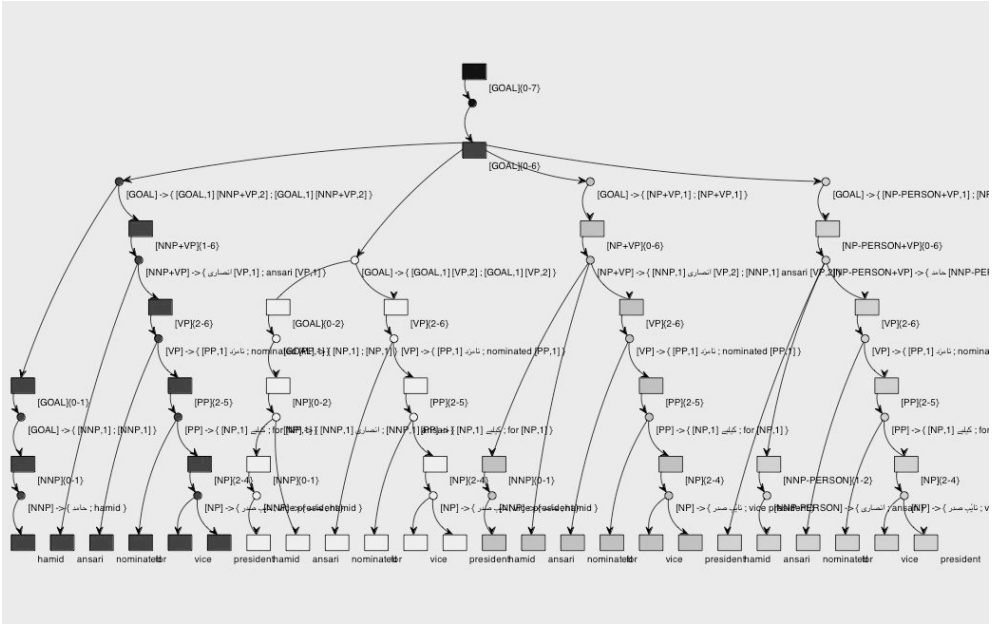

Figure 4. The visualization window for the hypergraph browser.

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

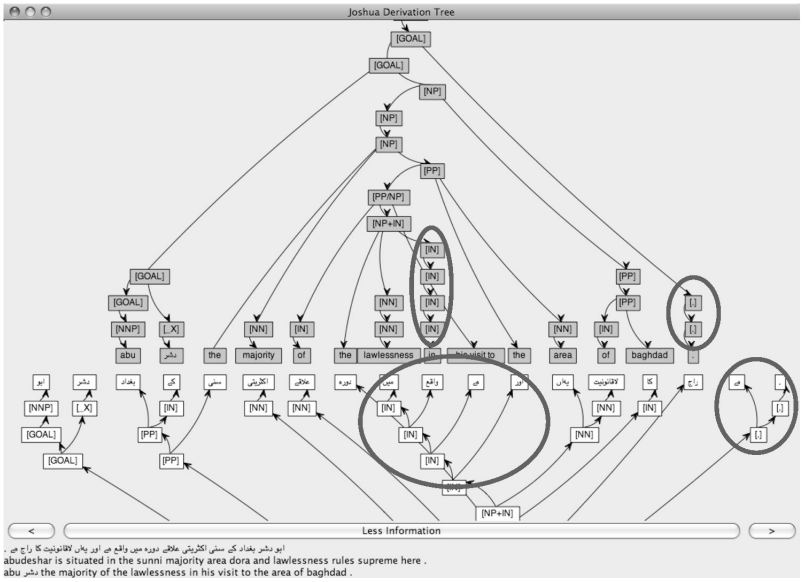

Figure 5. An example of bad production rules that parse pieces of the source sentence without producing any target-side output.

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.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 137–146

Continuous-Space Language Models for Statistical Machine Translation

Holger Schwenk

LIUM, University of Le Mans, France

Abstract

This paper describes an open-source implementation of the so-called continuous space language model and its application to statistical machine translation. The underlying idea of this approach is to attack the data sparseness problem by performing the language model probability estimation in a continuous space. The projection of the words and the probability estimation are both performed by a multi-layer neural network. This paper describes the theoretical background of the approach, efficient algorithms to handle the computational complexity, and gives implementation details and reports experimental results on a variety of tasks.

1. Introduction

Language models play an important role in statistical machine translation (SMT), i.e. phrase-based systems like Moses (Koehn et al., 2007) or hierarchical phrase-based systems like Joshua (Li et al., 2009). The classical equation of SMT shows this explicitly. Let us assume that we translate a French sentence f into English e:

$$e^* = \arg\max_e P(e|f) = \arg\max_e P(f|e)P(e) \quad (1)$$

the term $P(e|f)$ was explicitly rewritten with the Bayes rule in order to decompose the overall task into two components:

1. $P(f|e)$ gives the probability that f is a translation of e without necessarily paying much attention to the fact whether the generated sentence is well-formed or not;
2. $P(e)$ expresses the probability that the produced sentence is grammatically and semantically correct without looking at the source sentence.

 Corresponding author: holger.schwenk@lium.univ-lemans.fr
Cite as: Holger Schwenk. Continuous-Space Language Models for Statistical Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 137–146. ISBN 978-80-904175-4-0.
doi: 10.2478/v10108-010-0014-6.

The language model (LM) is responsible for the second task. It is clear that the importance of the LM increases when translating into morphologically rich languages. Despite this importance there seems to be only a limited amount of research in improved language modeling for SMT, and most of the state-of-the-art systems use the well-known n-gram back-off LMs, introduced more than 20 years ago. There is a large body of research on smoothing techniques, i.e. how to obtain probabilities for n-grams that were not observed in the training data, see for instance (Chen and Goodman, 1999) for an extensive comparison. In practice, modified Kneser-Ney smoothing is used in most of the cases. Today, there is a clear tendency to train those models on ever larger amounts of data, up to hundreds of billions of words. Several authors propose variants of the back-off n-gram approach to tackle the enormous computational and storage complexity during training and decoding, for instance (Federico and Cettolo, 2007; Brants et al., 2007; Talbot and Osborne, 2007).

In standard back-off n-gram language models words are represented in a discrete space, the vocabulary. This prevents "true interpolation" of the probabilities of unseen n-grams since a change in this word space can result in an arbitrary change of the n-gram probability. An alternative approach is based on a *continuous representation* of the words (Bengio et al., 2003; Schwenk, 2007). The basic idea is to convert the word indices to a continuous representation and to use a probability estimator operating in this space. Since the resulting distributions are smooth functions of the word representation, better generalization to unknown n-grams can be expected. This is still an n-gram approach, but an LM probability is available for any possible n-gram without the need to back-off to shorter contexts.

This continuous space LM was very successfully applied in large vocabulary continuous speech recognition (Schwenk, 2007) and references therein, and more recently in statistical machine translation (Schwenk et al., 2006a, 2007; Schwenk and Estève, 2008). In this paper we present an open-source implementation of this approach.

The paper is organized as follows. In the next section we first summarize the theoretical background of the CSLM. Section 3 presents implementation details of the toolkit and section 4 gives an overview on the performance of this approach.

2. Architecture of the continuous space language model

The architecture of the continuous space LM is shown in Figure 1. A standard fully-connected multi-layer perceptron is used. The inputs to the neural network are the indices of the $n-1$ previous words in the vocabulary $h_j = w_{j-n+1}, \ldots, w_{j-2}, w_{j-1}$ and the outputs are the posterior probabilities of *all* words of the vocabulary:

$$P(w_j = i|h_j) \qquad \forall i \in [1, N] \tag{2}$$

where N is the size of the vocabulary. The input uses the so-called 1-of-n coding, i.e. the ith word of the vocabulary is coded by setting the ith element of the vector to 1 and all the other elements to 0. The ith line of the $N \times P$ dimensional projection

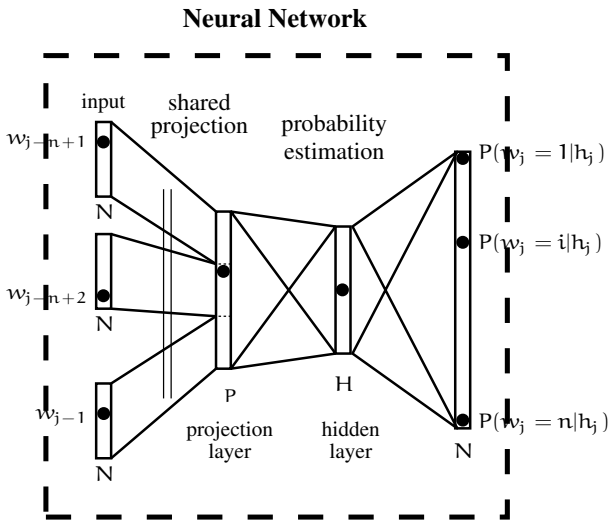

Figure 1. Architecture of the continuous space LM. h_j *denotes the context* $w_{j-n+1}, \ldots, w_{j-1}$. P *is the size of one projection and* H,N *is the size of the hidden and output layer respectively. When short-lists are used the size of the output layer is much smaller then the size of the vocabulary.*

matrix corresponds to the continuous representation of the ith word. Let us denote c_l these projections, d_j the hidden layer activities, o_i the outputs, p_i their softmax normalization, and m_{jl}, b_j, v_{ij} and k_i the hidden and output layer weights and the corresponding biases. Using these notations, the neural network performs the following operations:

$$d_j = \tanh\left(\sum_l m_{jl}\, c_l + b_j\right) \qquad j \in [1, H] \tag{3}$$

$$o_i = \sum_j v_{ij}\, d_j + k_i \qquad i \in [1, N] \tag{4}$$

$$p_i = e^{o_i} \,/\, \sum_{r=1}^{N} e^{o_r} \tag{5}$$

The value of the output neuron p_i corresponds directly to the probability $P(w_j = i|h_j)$. Training is performed with the standard back-propagation algorithm minimizing the

following error function:

$$E = \sum_{i=1}^{N} t_i \log p_i + \beta \left(\sum_{jl} m_{jl}^2 + \sum_{ij} v_{ij}^2 \right) \quad (6)$$

where t_i denotes the desired output, i.e. the probability should be 1.0 for the next word in the training sentence and 0.0 for all the other ones. The first part of this equation is the cross-entropy between the output and the target probability distributions, and the second part is a regularization term that aims to prevent the neural network from over-fitting the training data (weight decay). This is a well known technique when training neural networks by the back-propagation algorithm The parameter β has to be determined experimentally – in our experiments a value of 3^{-5} was used. Training is done using a resampling algorithm to weight multiple corpora.

It is well known that the outputs of a neural network trained by back-propagation converge to the posterior probabilities. Therefore, the neural network directly minimizes the perplexity on the training data. Note also that the gradient is back-propagated through the projection-layer, which means that the neural network learns the projection of the words onto the continuous space that is best for the probability estimation task.

The complexity to calculate one probability with this basic version of the continuous space LM is quite high due to the large output layer. To speed up the processing several improvements are implemented:

1. *Short-lists*: the neural network is only used to predict the LM probabilities of a subset of the whole vocabulary.
2. *Bunch mode*: several examples are propagated at once through the network. Instead of performing matrix/vector operations we have now matrix/matrix operations which can be heavily optimized on current CPU architectures.
3. *Grouping*: LM probabilities are not requested in a arbitrary order, but requests with the same context h_j are grouped together. By these means, only one forward pass through the neural network is needed.

The idea behind short-lists is to use the neural network to only predict the s most frequent words, s being much smaller than the size of the vocabulary. All words in the vocabulary are still considered at the input of the neural network. The LM probabilities of words in the short-list ($\hat{P}_N$) are calculated by the neural network and the LM probabilities of the remaining words ($\hat{P}_B$) are obtained from a standard 4-gram back-off LM:

$$\hat{P}(w_t|h_t) = \begin{cases} \hat{P}_N(w_t|h_t) P_S(h_t) & \text{if } w_t \in \text{short-list} \\ \hat{P}_B(w_t|h_t) & \text{else} \end{cases} \quad (7)$$

$$P_S(h_t) = \sum_{w \in \text{short-list}(h_t)} \hat{P}_B(w|h_t) \quad (8)$$

It can be considered that the neural network redistributes the probability mass of all the words in the short-list.[1] This probability mass is precalculated and stored with the models. A back-off technique is used if the probability mass for an input context is not directly available. In practice the short-list have a size of 8192 or 12288. This allows to cover about 90% of the LM requests.

It would be relatively straightforward to integrate the CSLM into the Moses decoder. This would however lead to very slow decoding time since the complexity to calculate an arbitrary n-gram probability is much higher for the CSLM than for a standard back-off LM (which in principle just does a table look-up). Several speed-up techniques are possible, but they are not yet implemented. It would be in particular necessary to group requests with the same context.

Instead we use the Moses decoder to create n-best lists which are then processed by a separate tool. This tool supports recalculation of the LM score using a CSLM or a higher-order back-off LM. Several CSLMs can be interpolated on the fly (it is not possible to merge multiple CSLMs into an bigger one like this is done for back-off n-gram models). This tool collects all the LM requests of an n-best lists and groups together requests with the same context h_j. By these means the number of forward passes through the neural network are drastically reduced. In addition, bunch mode is used.

3. Implementation details

The tool is written in C++ and all the routines are contained in the library `libcslm`. The current version is closely interfaced with the SRILM toolkit (Stolcke, 2002). We use in particular classes to load the vocabulary or back-off LMs for the short-list. Therefore, the SRILM toolkit must be installed. It is planed to also support IRSTLM and randomized language models in future versions. In addition, BLAS libraries[2] and the numerical optimization tool CONDOR are needed (see below). The CSLM toolkit provides the following executables:

- `cslm_train`: main tool to train CSLMs.
- `nbest_tool`: tool to process n-best lists (rescoring with a CSLM, recalculation of global scores, solution extraction, ...)
- `text2bin`: convert text files to a binary representation for faster loading

The tool kit can be downloaded from the following web-page: `http://liumtools.univ-lemans.fr`. Detailed documentation on the available options of the executables and examples are provided with the software.

The neural network routines of the library are generic and can be used for other applications of neural networks. Arbitrary multi-layer networks can be created, either

[1] Note that the sum of the probabilities of the words in the short-list for a given context is normalized $\sum_{w \in \text{short-list}} \hat{P}_N(w|h_t) = 1$.

[2] Basic Linear Algebra Subprograms, a commonly used library for matrix operations.

by composing them sequentially, i.e. the output of one layer is fed to the input of the following layer; or in parallel, i.e. the outputs of several layers are concatenated and build up a larger layer (this is used to compose the projection layer of the CSLM). Currently, only fully connected layers are implemented, i.e. each neuron in the input layer is connected with each neuron in the output layer. Training is performed with the standard back-propagation algorithm. Weight decay regularization is available (see equation 6). The neural network functions are implemented with the goal to achieve very fast training and recognition. For this high performance BLAS libraries are needed which take advantage of specifics of the processor to achieve fast matrix operations, e.g. SSE and multi-threading on Intel processors. We use the libraries MKL which are available from Intel for a very small fee,[3] but there are also freely available libraries, like ATLAS.[4] Speed comparison between different BLAS libraries were not performed.

The program `nbest_tool` is used to process Moses-style n-best list. Its main function is to calculate LM probabilities with the CSLM for all the hypotheses. In addition, it can be used to recalculate the global scores for a given set of parameters λ, sort the scores and extract the best hypotheses. This is used to reoptimize the system (see figure 2). It is planed to extend this tool with other operations on n-best lists, in particular MBR decoding.

The CSLM toolkit provides the program `text2bin` to convert the training texts into a binary representation. Basically, each word is replaced with its integer index in the word-list. It is more efficient to process this representation when using the resampling algorithm.

4. Experimental evaluation

The CSLM was initially applied to large vocabulary continuous speech recognition systems. It achieved significant reductions in the word error rate of up to 1% absolute for a large variety of languages and domains (Schwenk, 2007). Based on this success, the application to SMT was investigated, first in the framework of a system to translate texts from the European parliament proceedings (Schwenk et al., 2006b). The good results were then confirmed on a variety tasks, ranging from the resource-poor IWSLT evaluations (Schwenk et al., 2006a, 2007) to large NIST systems (Schwenk and Estève, 2008). These experiments are summarized in the following.

In all cases a two-pass approach was applied: first Moses was run using a 4-gram back-off LM and distinct 1000-best lists were created. Those 1000-best lists were then rescored with the program `nbest_tool`. The LM probabilities calculated by the CSLM can be used to replace those of the standard back-off LM or added as an additional feature function. In both cases we performed a minimum error training of the coeffi-

[3] Math Kernel Library, `http://software.intel.com/en-us/intel-mkl/`

[4] `http://math-atlas.sourceforge.net/`

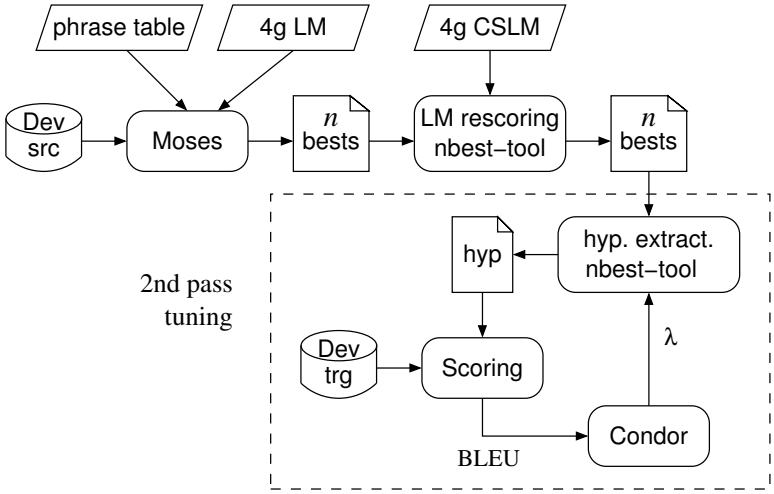

Figure 2. Two pass decoding architecture. First distinct n*-best lists are created with Moses using a 4-gram back-off LM. These* n*-best lists are rescored with the CSLM. The resulting* n*-best lists are then used to retune the coefficients of the feature functions (2nd pass tuning). For this the freely numerical optimization tool* Condor *is used. It iteratively provides a set of parameters* λ *which are used to recalculate the global score and to extract the current best hypotheses. This is done with the program nbest-tool. Those hypotheses are evaluated against the reference translations. The BLEU score is used by the* Condor *tool to propose a new set of parameters* λ*, until convergence.*

cients of the feature functions, using the freely available numerical optimization tool CONDOR (Berghen and Bersini, 2005). CONDOR performs a Powell-style numerical optimization. The two-pass architecture is summarized in figure 2. Examples of tuning with CONDOR and supporting scripts are provided with the CSLM toolkit.

The goal of the IWSLT BTEC[5] task is to translate typical expressions from the travel domain, mainly between Asian languages and English. There are about 200 to 400 thousand words of bitexts available to train the translation models, in function of the source language. The LM is trained on the English side of those texts. Given the particular domain of this task, adding texts from other sources, e.g. newspaper texts from LDC's Gigaword corpora, has only a small impact. Therefore, it should be promising to apply the CSLM to this task since it is expected to take better advantage of the limited amount of in-domain LM training data. This could be confirmed by the ex-

[5]Basic Travel Expression Corpus

perimental results using standard phrase-based as well as n-gram based SMT systems (Schwenk et al., 2006a). Those results are summarized in Table 1.

	Phrase-based		N-gram-based	
	Ref.	CSLM	Ref.	CSLM
Zh/En	19.74	21.01	20.34	21.16
Ja/En	15.11	15.73	16.14	16.35
Ar/En	23.72	24.86	23.83	23.70
It/En	35.55	37.41	35.95	37.65

Table 1. BLEU scores on the IWSLT 2006 test data, 4-gram back-off versus CSLM for phrase-based and n-gram-based translation systems.

It is also interesting to use the CSLM in conjunction with an n-gram based SMT system to improve the *translation model*. This approach basically uses a language model on bilingual tuples (Mariño et al., 2006), resulting in the same data sparseness problems as for the model on the target language. Practically, we just need to apply the CSLM to change the corresponding feature function in the n-best list. Table 2 presents those results for the Italian/English language pair (Schwenk et al., 2007). The continuous space language model achieved a gain of 1 point BLEU while the continuous space translation model only brought a little more than 0.2 BLEU points. However, both models together achieved a gain of 1.5 points BLEU.

	Back-off TM+LM	neural TM	neural LM	**neural TM+LM**
It/En	36.97	37.21	38.04	**38.50**

Table 2. BLEU scores on IWSLT 2006 test data for the combination of a neural translation model (TM) and a neural language model (LM).

Finally, the method was applied to a task for which very large amounts of LM training data are available, namely the NIST evaluation tasks. The goal of this evaluations is to translate newspaper and web texts from Chinese and Arabic into English. In addition to the English side of about 200 million words of bitexts, the LDC collection of newspaper texts can be considered to be close to in-domain. Those texts are known as the Gigaword corpus and they amount to more than 3 billion words. It is of course impossible to train a neural network on so many examples and this would be even a bad idea: large collections like the AFP texts would dominate the smaller but important text sources. It is well known that it is better to build separate back-off LMs

on each text source, to determine a coefficient for each one and to interpolate them together. Loosely inspired by this technique, a resampling method was used to train the neural network. At each epoch of back-propagation training, we resample randomly a subset of each corpus. The resampling coefficient is chosen to take all of the small and important corpora, and smaller subsets of the other "*background corpora*". More details of this procedure are given in (Schwenk, 2007) and references therein.

The Arabic English system was optimized on the NIST'06 test set and tested on the NIST'08 data. The reference systems achieves a BLEU score of 47.02 using a very large back-off LM that was trained by keeping all the 4-grams that appear in more than 3.3 million words of texts. Applying the CSLM, again by rescoring distinct 1000-best lists, improves the BLEU score to 47.90. The final system achieved a very good ranking in the 2009 NIST evaluation.[6]

5. Conclusion

This paper described an open-source implementation of a continuous space language model for SMT. The integration of the CSLM into the translation process is performed by a two-pass approach: first n-best lists are generated which are then rescored using a provided tool. This approach can be used with different types of machine translation systems, as long as they are able to produce n-best lists that contain the scores of all the feature functions. During the last years, continuous space LMs were successfully applied to a variety of SMT systems. Improvements of the BLEU scores of more than 1 point BLEU were observed and, in general, this also lead to better human judgments.

The CSLM toolkit is meant to be a starting point for ongoing research in the field of estimating probabilities in the continuous domain. Future versions may include factored representations of the words, support for lattices instead of n-best lists and an application of the approach to the translation model.

6. Acknowledgments

This work has been partially funded by the French Government under the project Instar (ANR JCJC06 143038) and the European project FP7 EuromatrixPlus.

[6] http://www.itl.nist.gov/iad/mig/tests/mt/2009/ResultsRelease/currentArabic.html

Bibliography

Bengio, Yoshua, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(2):1137–1155, 2003.

Berghen, Frank Vanden and Hugues Bersini. CONDOR, a new parallel, constrained extension of powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. *Journal of Computational and Applied Mathematics*, 181:157–175, 2005.

Brants, Thorsten, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Empirical Methods in Natural Language Processing*, pages 858–867, 2007.

Chen, Stanley F. and Joshua T. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

Federico, Marcello and Maura Cettolo. Efficient handling of n-gram language models for statistical machine translation. In *Second ACL Workshop on Statistical Machine Translation*, pages 88–95, 2007.

Koehn et al., Philipp. Moses: Open source toolkit for statistical machine translation. In *ACL, demonstration session*, 2007.

Li, Zhifei, Chris Callison-Burch, Sanjeev Khudanpur, and Wren Thornton. Joshua: Open source, parsing-based machine translation. *The Prague Bulletin of Mathematical Linguistics*, (91), 2009.

Mariño, J.B., R.E. Banchs, J.M. Crego, A. de Gispert, P. Lambert, J.A.R. Fonollosa, and M. R. Costa-jussà. Bilingual n-gram statistical machine translation. *Computational Linguistics*, 32(4):527–549, December 2006.

Schwenk, Holger. Continuous space language models. *Computer Speech & Language*, 21: 492–518, 2007.

Schwenk, Holger and Yannick Estève. Data selection and smoothing in an open-source system for the 2008 NIST machine translation evaluation. In *Interspeech*, pages 2727–2730, 2008.

Schwenk, Holger, Marta R. Costa-jussà, and José A. R. Fonollosa. Continuous space language models for the IWSLT 2006 task. In *International Workshop on Spoken Language Translation*, pages 166–173, November 2006a.

Schwenk, Holger, Daniel Déchelotte, and Jean-Luc Gauvain. Continuous space language models for statistical machine translation. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 723–730, 2006b.

Schwenk, Holger, Marta R. Costa-jussà, and José A. R. Fonollosa. Smooth bilingual n-gram translation. In *Empirical Methods in Natural Language Processing*, pages 430–438, 2007.

Stolcke, Andreas. SRILM - an extensible language modeling toolkit. In *International Conference on Speech and Language Processing*, pages II: 901–904, 2002.

Talbot, David and Miles Osborne. Randomised language modelling for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 512–519, 2007. URL `http://www.aclweb.org/anthology/P07-1065`.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 147–155

MANY

Open Source Machine Translation System Combination

Loïc Barrault

LIUM, University of Le Mans

Abstract

This paper describes a *push-the-button* MT system combination toolkit. The combination is based on the creation of a lattice made on several confusion networks (CN) connected together. This lattice is then decoded with a token-pass decoder to provide the best and/or n-best outputs. Each CN is built using a modified version of the TERp tool. The toolkit is made of several scripts along a core program developed in Java. It is totally configurable and the parameters can be tuned quite easily.

1. Introduction

Machine translation (MT) system combination has taken a great importance these past few years. This is mainly due to the fact that single systems achieved good performance and the possibility of taking the most of their complementarity in a system combination framework is very attractive. Many techniques can be used for system combination. One concerns hypothesis selection using nbest list reranking based on various features as described in (Hildebrand and Vogel, 2009). Another approach is to consider source text and systems outputs as bitext and train a new SMT system on these data (Chen et al., 2009).

In this paper, a system combination based on confusion network (CN) is described. This approach is not new, and numerous publications are available on that subject, see for example, (Rosti et al., 2007); (Shen et al., 2008); (Karakos et al., 2008) and (Leusch et al., 2009). Such an approach is presented in Figure 1. The protocol can be decomposed into three steps :

Corresponding author: loic.barrault@gmail.com
Cite as: Loïc Barrault. MANY: Open Source Machine Translation System Combination. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 147–155. ISBN 978-80-904175-4-0.
doi: 10.2478/v10108-010-0001-y.

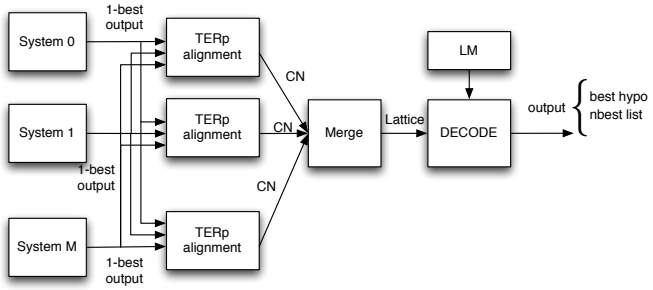

Figure 1. MT system combination. Each 1-best outputs are aligned to create as many Confusion Networks which are connected together to form a lattice. This lattice is then decoded with a token-pass decoder using a Language Model to produce 1-best and/or n-best hypotheses.

1. 1-best hypotheses from all M systems are aligned in order to build confusion networks.
2. All confusion networks are connected into a single lattice.
3. A language model is used to decode the resulting lattice and the best hypothesis is generated.

Section 2.1 describes the alignment process and in particular the new features added to TERp in order to be enable alignment of an hypothesis against a CN. The decoder is presented in section 3. Some example results obtained at the IWSLT'09 evaluation campaign are given in section 5. Finally, a description of the toolkit is given in section 6.

2. Hypotheses alignment and confusion network generation

The goal of this step is to put the words provided by different systems in competition with each other inside a confusion network (Mangu et al., 1999).

For each segment, the best hypotheses of $M - 1$ systems are aligned against the last one used as backbone. A modified version of the TERp tool (Snover et al., 2009a) (Snover et al., 2009b) is used to generate a confusion network (see section 2.1 for details). This is done by incrementally adding the hypotheses to the CN. The hypotheses are added to the backbone beginning with the nearest (in terms of TER) and ending with the more distant one. This differs from the result of (Rosti et al., 2007) where the nearest hypothesis is computed at each step, which is supposed to be better. M confusion networks are generated in this way. Then all the confusion networks are connected into a single lattice by adding a first and last node. The probability of the

first arcs (later named priors) must reflect how well such system provides a well structured hypothesis.

2.1. Modified TERp

The modified TERp is based on TERp v0.1 and is written in Java. Some classes have been modified and new ones were created to add some functionalities such as alignment between a sentence and a confusion network. This has been done by modifying the data structure and extending some heuristic to find better alignment.

When using *relaxed* constraints with TERp, the shift heuristics allow a block of words to be moved if it matches (or is a paraphrase of) another block of words somewhere else. Shifts are also allowed when a stem or synonym is found somewhere else.

When considering confusion networks, the same heuristics are applied except that the block of words must match (be a paraphrase, synonyms or stem of) one of the sequence of words represented in the CN. An example of such a case is presented in figure 2. In figure 2, we can notice that the paraphrase *the dinner* / *supper* allow

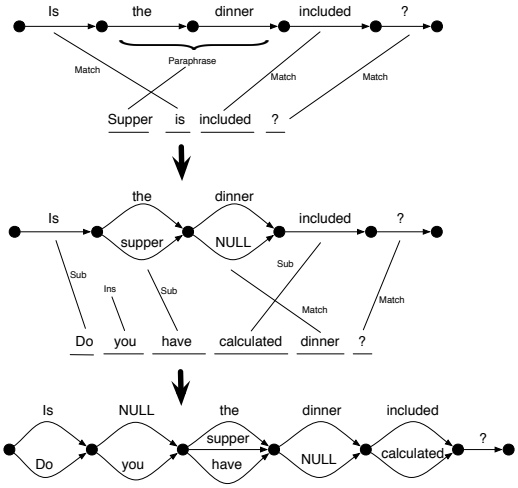

Figure 2. Incremental alignment with TERp resulting in a confusion network.

a switch of block of word. However, the word *supper* is aligned with the word *the* because no rule is used in order to make inside-paraphrase word alignment, yet ! (see section 6.4 for future features).

In addition to the confusion network generation, the possibility of using scores on words has been added, which can be very useful during the decoding. For the moment, these scores must be computed separately from MANY. The underlying idea is to provide an option to include confidence measure at word level, though it can be computed at any level (see for example, (Ueffing and Ney, 2005)). In this version of the software, the scores are equal to the priors of the systems. However, these values can be modified in the configuration file.

3. Decoding

The decoder is based on the token pass decoding algorithm (see for example (Young et al., 1989)). The principle of this decoder is to propagate tokens over the lattice and accumulate various scores into a global score for each hypotheses.

The scores used to evaluate the hypotheses are the following :

- the system score : this replace the score of the translation model. Until now, the words given by all systems have the same probability which are equal to their priors, but any confidence measure can be used at this step.
- the language model (LM) probability.
- a fudge factor to balance the probabilities provided in the lattice with regard to those given by the language model.
- a *null-arc* penalty : this penalty avoids to always go through *null-arcs* encountered in the lattice.
- a length penalty : this score helps to generate correctly sized hypotheses.

The probabilities computed in the decoder can be expressed as follow :

$$\begin{aligned} \log(P_W) &= \sum_{n=0}^{Len(W)} [\log(P_{ws}(n)) + \alpha P_{lm}(n)] \\ &\quad + Len_{pen}(W) + Null_{pen}(W) \end{aligned} \tag{1}$$

where $Len(W)$ is the length of the hypothesis, $P_{ws}(n)$ is the score of the n^{th} word in the lattice, $P_{lm}(n)$ is its LM probability, α is the fudge factor, $Len_{pen}(W)$ is the length penalty of the word sequence and $Null_{pen}(W)$ is the penalty associated with the number of null-arcs crossed to obtain the hypothesis.

At the beginning, only one token is created at the first node of the lattice. Then this token spread over the consecutive nodes, accumulating the score on the arc it crosses, the language model probability of the word sequence generated so far and null or length penalty if applicable. The number of tokens can increase really quickly to cover the whole lattice, and, in order to keep it tractable, only the N_{max} best tokens are kept (the others are discarded), where N_{max} can be configured in the configuration file. Other methods to restrict the number of tokens (like pruning based on score or other heuristics) can easily be implemented in this software, but this is not done already.

3.1. Technical details about the token pass decoder

This software is based on the Sphinx4 library and is highly configurable (Walker et al., 2004). The maximum number of tokens being considered during decoding, the fudge factor, the null-arc penalty and the length penalty can all be set within the xml configuration file. This is useful for tuning (see the config file generator description in section 6.2).

The probabilities which are manipulated within the decoder are all obtained from the LogMath class which ensures the consistency of the values.

3.2. Language model

There are two ways of loading a LM with this software.

The first solution is to use the LargeTrigramModel class, but as its name tells us, only a 3-gram model can be loaded with this class.

The second and easiest way is to use a language model hosted on a lm-server. This kind of LM can be accessed via the LanguageModelOnServer class which is based on the generic LanguageModel class from the Sphinx4 library. This allows us to load a n-gram LM with n higher than 3, which is not possible with a standard LM class in Sphinx4 yet (it is currently being done).

In addition, the Dictionary interface has been extended in order to be able to load a simple dictionary containing all the words known by the LM (no need to know the different pronunciations of each words in this case).

As the language model interface is also written in java and is using the Sphinx4 library, one could easily write a new class to load a LM in a proprietary file format.

4. Tuning

There is a lot of parameters which can be tuned in MANY. The edit costs of the modified TERp, the prior costs of each systems in the lattice, the fudge, null-arc penalty and length penalty for the decoder. This can easily been done by generating configuration files (with the help of *genSphinxConfig.pl*, see section 6.3). Parameters for modified TERp, for the decoder and systems weights are currently tuned together. The separate tuning of TERp and decoder parameters is an ongoing work, and I could not say whether it is preferable or not yet.

Any method can then be used to provide new values for these parameters. As an example, we are using Condor (Berghen and Bersini, 2005) to optimize those parameters.

5. Some example results

MANY software has been used for the IWSLT'09 evaluation campaign. Table 1 presents the results obtained with this approach. The SMT system is based on MOSES,

the SPE system corresponds to a rule-based system from SYSTRAN whose outputs have been corrected by a SMT system and the Hierarchical is based on Joshua.

Systems	Arabic/English		Chinese/English	
	Dev7	Test09	Dev7	Test09
SMT CSLM	54.75	50.35	41.71	36.04
SPE CSLM	48.13	-	41.23	38.53
Hierarchical	54.00	49.06	39.78	31.89
SMT CSLM + SPE CSLM			42.55	40.14
+ tuning			43.06	39.46
SMT CSLM + Hier.	55.89	50.86		
+ tuning	57.01	51.74		

Table 1. Results of system combination on Dev7 (development) corpus and Test09, the official test corpus of IWSLT'09 evaluation campaign.

In these task, the system combination approach yielded +1.39 BLEU on Ar/En and +1.7 BLEU on Zh/En. One observation is that tuning parameters did not provided better results for Zh/En.

6. Software description

The software is available at the following address :

http://www-lium.univ-lemans.fr/~barrault/MANY

6.1. Data

The software takes several files as input (which are supposed to be synchronized[1]) containing the 1-best hypothesis of all systems, one sentence per line. These hypotheses can contain foreign words if no translation have been found for them, and they will be considered as unknown words during the decoding step.

6.2. Configuration file

The configuration file is an xml file similar to those used with Sphinx4.

```
<component  name="decoder" type="edu.loic.decoder.TokenPassDecoder">
<property name="dictionary" value="dictionary"/>
<property name="logMath" value="logMath"/>
<property name="logLevel"     value="INFO"/>
```

[1] *i.e.* each n^{th} line is the translation of the same source sentence

```
<property name="lmonserver" value="lmonserver"/>
<property name="fudge" value="0.2"/> <!-- This value is multiplied by 10 in the software -->
<property name="null_penalty" value="0.3"/>
<property name="length_penalty" value="0.5"/> <!-- This value is multiplied by 10 in the software -->
</component>
```

This part allows us to configure the decoder parameters such and more particularly the fudge factor, the null-arc penalty and the length penalty.

```
<component  name="lmonserver" type="edu.cmu.sphinx.linguist.language.ngram.LanguageModelOnServer">
<property name="lmserverport" value="1234"/>
<property name="lmserverhost" value="machine1"/>
<property name="maxDepth" value="4"/>
<property name="logMath" value="logMath"/>
</component>
```

This part configures the LM class which will connect to the lm-server hosted on *machine1* on port "1234". The *maxDepth* field correspond to the depth of the LM loaded on the server.

```
<component name="MANY" type="edu.lium.mt.MANY">
<property name="decoder" value="decoder"/>
<property name="terp" value="terp"/>
<property name="output" value="output.many"/>
<property name="priors" value="4.0e-01 4.0e-01 2.0e-01"/>
<property name="hypotheses" value="hyp0.id hyp1.id hyp2.id" />
<property name="hyps_scores" value="hyp0_sc.id hyp1_sc.id hyp2_sc.id" />
<property name="costs" value="1.0  1.0   1.0   1.0   1.0   0.0   1.0" />
<!--                        del  stem   syn   ins  sub   match shift-->
<property name="terpParams"     value="terp.params"/>
<property name="wordnet"     value="/opt/mt/WordNet-3.0/dict/"/>
<property name="shift_word_stop_list" value="/opt/mt/terp/terp.v1/data/shift_word_stop_list.txt"/>
<property name="paraphrases" value="/opt/mt/terp/terp.v1/data/phrases.db"/>
</component>
```

This part is the core part. It configures the various files to combine, the costs for TERp, the location of WordNet and the paraphrases table (also for TERp). The *priors* can be set here and are used in the lattice.

6.3. Scripts

The main script is called *Many.sh*. Some parameters have to be set inside this script in order to run a system combination experiments. The reader should refer to the *readme* file provided with the software.

Each input sentence (as well as the corresponding word scores) must have an id which is of the following form : [set][doc.##][sent] The shell script *add_id.sh* is in charge of adding such an id to the input data (called in the Many.sh script).

The perl script *genSphinxConfig.pl* is used to generate a new config file with specific values. This is very useful for generating a new config file with parameters estimated by a certain optimization procedure.

6.4. Future features

Several features are planned to be added into MANY. One is the possibility of exploring all shifts which do not decrease the alignment score instead of using heuris-

tics. This has been done by (Rosti et al., 2009) and provided good results (even though the increasing time of processing was not indicated).

Another feature would be the intra-paraphrase word alignment. Like is presented in figure 2, when a paraphrase is found, it appears that the word alignment inside that paraphrase is not always the best. In that example, (*supper* is aligned with *the* instead of *dinner*, which would be better. This could be easily added using a specific alignment model.

As mentioned before, the load of a n-gram (whatever is n) language model has to be added. In some cases, that can be faster than using a LM server.

An alternative to the token pass decoder would be the use of Minimum Bayesian Risk decoder applied on the final lattice (MBR-Lattice) like described in (Tromble et al., 2008)

7. Discussion

One might notice that the performance of a system combination is highly dependent of the input hypotheses (in terms of number of hypotheses, complementarity of the systems which provide them, and of course quality), the parameters of the alignment module and the language model used to decode the lattice. The tuning of all parameters plays consequently a big role in the quality of this kind of approach. As an example, in (Rosti et al., 2009), after the creation of the lattice, three iterations of tuning have been done in order to obtain good results. This kind of tuning procedure is not currently implemented in that software, but it is a very important step which should not be underestimated.

8. Conclusion

This paper presents a machine translation system combination software, MANY, based on the decoding of a lattice made of several confusion networks connected together. The software is written in java and is composed of a modified version of TERp software and a decoder based on Sphinx4 library. This software, which is easily extensible and highly configurable, obtained good results when used during the IWSLT'09 evaluation campaign.

Bibliography

Berghen, Frank Vanden and Hugues Bersini. CONDOR, a new parallel, constrained extension of powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. *Journal of Computational and Applied Mathematics*, 181:157–175, September 2005.

Chen, Yu, Michael Jellinghaus, Andreas Eisele, Yi Chang, Sabine Hunsicker, Silke Theison, Christian Federmann, and Hans Uszkoreit. Combining multi-engine translations with moses. In *Workshop on Statistical Machine Translation*, pages 42–46, Athens, Greece, March 2009.

Hildebrand, Almut Silja and Stephan Vogel. CMU system combination for WMT'09. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 47–50, Athens, Greece, March 2009.

Karakos, Damianos, Jason Eisner, Sanjeev Khudanpur, and Markus Dreyer. Machine translation system combination using ITG-based alignments. In *46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies.*, pages 81–84, Columbus, Ohio, USA, June 16-17 2008.

Leusch, G., E. Matusov, and H. Ney. The RWTH system combination system for WMT 2009. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 61–65, Athens, Greece, March 30-31 2009.

Mangu, L., E. Brill, and A. Stolcke. Finding consensus among words : Lattice-based word error minimization. In *European Conference on Speech Communication and Technology, Interspeech*, volume I, pages 495–498, 1999.

Rosti, A.-V.I., S. Matsoukas, and R. Schwartz. Improved word-level system combination for machine translation. In *Association for Computational Linguistics*, pages 312–319, 2007.

Rosti, A.-V.I., B. Zhang, S. Matsoukas, , and R. Schwartz. Incremental hypothesis alignment with flexible matching for building confusion networks: BBN system description for WMT09 system combination task. In *EACL/WMT*, pages 61–65, 2009.

Shen, Wade, Brian Delaney, Tim Anderson, and Ray Slyh. The MIT-LL/AFRL IWSLT-2008 MT System. In *International Workshop on Spoken Language Translation*, Hawaii, U.S.A, 69–76 2008.

Snover, Matthew, Nitin Madnani, Bonnie Dorr, and Richard Schwartz. Fluency, adequacy, or HTER? exploring different human judgments with a tunable MT metric. In *Workshop on Statistical Machine Translation*, Athens, Greece, March 2009a.

Snover, Matthew, Nitin Madnani, Bonnie Dorr, and Richard Schwartz. TER-Plus: Paraphrase, semantic, and alignment enhancements to translation edit rate. *Machine Translation Journal*, 2009b.

Tromble, Roy W., Shankar Kumar, Franz Och, and Wolfgang Macherey. Lattice Minimum Bayes-Risk decoding for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, pages 620–629, Honolulu, Oct. 2008.

Ueffing, Nicola and Hermann Ney. Word-level confidence estimation for machine translation using phrase-based translation models. In *International Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 763–770, Morristown, NJ, USA, 2005. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220575.1220671.

Walker, Wille, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical Report TR-2004-139l, Sun Microsystems Laboratories, Novembre 2004.

Young, S. J., N. H. Russell, and J. H. S. Thornton. Token passing : a simple conceptual model for connected speech recognition systems. Technical report, Cambridge University Engineering Department, July 1989.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 157–166

Hierarchical Phrase-Based Grammar Extraction in Joshua

Suffix Arrays and Prefix Trees

Lane Schwartz[a], Chris Callison-Burch[b]

[a] University of Minnesota, Minneapolis
[b] Center for Language and Speech Processing, Johns Hopkins University, Baltimore

Abstract

While example-based machine translation has long used corpus information at run-time, statistical phrase-based approaches typically include a preprocessing stage where an aligned parallel corpus is split into phrases, and parameter values are calculated for each phrase using simple relative frequency estimates. This paper describes an open source implementation of the crucial algorithms presented in (Lopez, 2008) which allow direct run-time calculation of SCFG translation rules in Joshua.

1. Introduction

A significant amount of the recent research in statistical machine translation has focused on modeling translation based on contiguous strings of words, called *phrases*, in the source language and corresponding phrases in the target language. Phrase-based translation (Och et al., 1999; Koehn et al., 2003; Marcu and Wong, 2002; Och and Ney, 2004) have proved to be very successful, and many state-of-the-art machine translation systems are based on these approaches.

A critical component in phrase-based translation is the estimation of a translation model from a word-aligned parallel text. A phrase table containing the source phrases, their target translations and their associated probabilities that is typically extracted in a preprocessing stage before decoding a test set (Koehn et al., 2003; Kumar et al., 2006). An example of this preprocessing approach is found in the training scripts provided as part of the open source phrase-based Moses toolkit (Koehn et al., 2007). Hierarchical phrase-based translation (Chiang, 2005) extends phrase-based transla-

Corresponding author: lane@cs.umn.edu
Cite as: Lane Schwartz, Chris Callison-Burch. Hierarchical Phrase-Based Grammar Extraction in Joshua: Suffix Arrays and Prefix Trees. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 157–166. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0013-7.

tion by allowing phrases with gaps, modeled as a synchronous context-free grammar (SCFG). The original Hiero implementation (Chiang, 2007) trains its SFCG translation model in a similar preprocessing stage.

By contrast, example-based machine translation (EBMT) approaches (Nagao, 1981; Sato and Nagao, 1990; Somers, 2003) are notable for their use of aligned parallel corpora at run time. EBMT research has successfully explored how various efficient data structures for pattern matching (Brown, 2004) can be leveraged to allow the decoder to access at decode-time portions of the training text that are most relevant to the text currently being translated. The Cunei machine translation toolkit (Phillips and Brown, 2009) is an open source, statistical EBMT system that follows this approach.

Suffix arrays are compact data structures which allow efficient pattern matching to be performed over all text in a corpus (Manber and Myers, 1990). Callison-Burch et al. (2005) and Zhang and Vogel (2005) showed that suffix arrays can be adapted to allow phrase-based translation to calculate translation options for the translation model at run-time. A subsample of occurrences of given source phrase are used to calculate translation probabilities. By accessing the target corpus and word alignment data, the phrasal translations and their associated model parameters can be calculated at run-time. Lopez (2007) showed that hierarchical phrases can also by obtained at run time using a suffix array.

This article reports on an implementation of the basic techniques described in Lopez (2008) that was incoporated into the open source machine translation system Joshua (Li et al., 2009). The implementation described here enables users of Joshua to begin translating sentences using an aligned parallel corpus without having to extract an SCFG before decoding begins. The advantages of using this implementation are that any input sentence can be decoded (making it appropriate for live demos or real world use), and that the data structures require much less disk space than full phrase tables. This comes at the cost of slower running time for the decoder itself, since phrase translations have to be calculated on the fly.

2. Related Work

While example-based machine translation has long used corpus information at run-time, statistical phrase-based approaches typically include a preprocessing stage where an aligned parallel corpus is split into phrases, and parameter values are calculated for each phrase using simple relative frequency estimates. The phrase-based decoders Pharaoh (Koehn, 2004) and Moses (Koehn et al., 2007) take this approach, providing users with scripts to estimate a translation model from a sentence-aligned parallel corpus. Similarly, Hiero (Chiang, 2007) and the syntax-augmented machine translation (SAMT) system (Zollmann and Venugopal, 2006) both require a preprocessing stage to extract a SCFG translation model. Recent work in Moses (Huang and Koehn, p.c.) provides similar functionality for extracting an SCFG-based translation model during a preprocessing stage.

Callison-Burch et al. (2005), Zhang and Vogel (2005), and Lopez (2008) all describe implementations of traditional phrase-based models extended to take advantage of suffix array data structures to extract phrase translation options at run-time. However, functional open source implementations of these have yet to be made available. Preliminary work has investigated integrating these techniques into Moses, but this work is not complete.[1]

Lopez (2008) provides a fast implementation of SCFG grammar extraction for Hiero which uses suffix arrays. This implementation allows Hiero to use an aligned parallel corpus at run-time in lieu of a pre-extracted SCFG. However, this implementation is not available as open source software due to intellectual property restrictions imposed by the University of Maryland. Cunei (Phillips and Brown, 2009)[2] is a statistical open source EBMT system that uses suffix arrays to extract relevant phrase pairs from an aligned parallel corpus at run-time.

3. Implementation: Data Structures and Algorithms

To extract hierarchical translation rules at run-time, the decoder must have access to the aligned parallel corpus. Internally, Joshua treats all source and target words as 32-bit integers. Each unique string that is encountered is assigned a unique integer. A hash map maintains the mapping from string to integer, while a corresponding list of strings maintains the mapping from integer back to string. Together these data structures comprise the **symbol table**.

A corpus can be considered a simple list whose size is equal to the number of words in the corpus. Using this approach with the symbol table, Joshua stores the source corpus as an integer array. An auxiliary array, with size equal to the number of sentences in the source corpus, is maintained. Elements in this auxiliary sentence array indicate the corpus index where each sentence begins. The target corpus is likewise represented by a **corpus array** and corresponding sentence array.

Once the source and target corpus arrays are available, the corresponding **suffix array**s can be constructed. Given a corpus array c and a symbol table, a second array is created of equal size to the corpus array. This array s is initialized such that $s[x] = x$. Where each integer in c represents a word string, each integer in s represents an index into c. The contents of s are sorted, using the element comparison function defined in Figure 1. After sorting, the indices of all instances in the corpus of any given phrase are located in a contiguous segment in the suffix array s.

While a phrase-based decoder can simply look up any required phrase in a suffix array, hierarchical decoders must deal with discontinuous phrases that include gaps. To deal with such phrases, Lopez (2008) defines an incremental algorithm for

[1] Much of this preliminary work was conducted by Chris Callison-Burch, Andreas Eisele, Juri Ganitkevitch, and Adam Lopez at the Second Machine Translation Marathon in 2008.

[2] `http://sourceforge.net/projects/cunei`

```
1: function Compare_Elements(index1, index2, max, corpusEnd)
2:     for i = 0; i < max; i++ do
3:         if index1 + i < corpusEnd and index2 + i > corpusEnd then
4:             return 1
5:         else if index2 + i < corpusEnd and index1 + i > corpusEnd then
6:             return −1
7:         else if corpus[index1 + i] is lexicographically < corpus[index2 + i] then
8:             return −1
9:         else if corpus[index1 + i] is lexicographically > corpus[index2 + i] then
10:            return 1
11:        end if
12:    end for
13:    return 0
14: end function
```

Figure 1. During suffix array creation, the contents of a corpus array are sorted using the element comparison function Compare_Elements

constructing a specialized trie (Fredkin, 1960) to represent the SCFG translation grammar. Given a source sentence, this algorithm constructs a **prefix tree with suffix links** by first examining all possible contiguous source phrases, and uses the source suffix array to look up translations for contiguous phrases. Hierarchical phrases that consist of a contiguous phrase preceded or followed by a single nonterminal X can be constructed directly from the corresponding contiguous phrase. In this manner, the tree is gradually constructed into a grammar containing contiguous phrases and simple hierarchical phrases.

More complex hierarchical phrases are constructed using the Query_Intersect function in Figure 2. This function takes two smaller phrases $a\alpha$ and αb (a and b represent single words and α represents a sequence), along with the list of indices where these phrases are located. These two lists can be efficiently processed to determine the locations where the two phrases intersect to form the more complex phrase $a\alpha b$. In this way, all source hierarchical phrases can be located.

Each node in the prefix tree corresponds to a unique source phrase. Each node stores the complete list of all indices in the source corpus where that node's phrase occurs. These locations are used in conjunction with the target corpus array and the word alignment data to construct SCFG translation rules.

Ideally, once translation rules have been extracted for a given source phrase, those rules would be stored and not calculated again. Memory constraints typically dictate that not all rules are stored. Rather than storing the translation rules for a given source phrase at the corresponding node in the prefix tree, a single least-recently-used (LRU)

Algorithm QUERY_INTERSECT
Input: Sorted list of corpus locations matching source language pattern $a\alpha$: $M_{a\alpha}$
Input: Sorted list of corpus locations matching source language pattern αb: $M_{\alpha b}$

function QUERY_INTERSECT($M_{a\alpha}$, $M_{\alpha b}$)
  $M_{a\alpha b} \leftarrow \emptyset$ ▷ Result list is initially empty
  $I \leftarrow |M_{a\alpha}|$ ▷ Number of instances of pattern $a\alpha$ in the source corpus
  $J \leftarrow |M_{\alpha b}|$ ▷ Number of instances of pattern αb in the source corpus
  $j \leftarrow 0$
  $i \leftarrow 0$
  while $i < I$ and $j < J$ **do**
    ▷ Ignore elements in $M_{\alpha b}$ that are
    ▷ too distant to intersect with $M_{a\alpha}[i]$
    while $j < J$ and $M_{a\alpha}[i] \ddot{>} M_{\alpha b}[j]$ **do**
      $j \leftarrow j + 1$
    end while
    ▷ Verify that the corpus index
    ▷ for the first terminal symbol
    $k \leftarrow i$ ▷ in pattern αb is the same
    while $M_{\alpha b}[i]_{,1} = M_{\alpha b}[k]_{,1}$ **do** ▷ for locations $M_{\alpha b}[i]$ and $M_{\alpha b}[k]$
      $\ell \leftarrow j$
      while $\ell < J$ and not $M_{a\alpha}[i] \ddot{<} M_{\alpha b}[\ell]$ **do**
        if $M_{a\alpha}[i] \ddot{=} M_{\alpha b}[\ell]$ **then**
          Intersect $M_{a\alpha}[i]$ with $M_{\alpha b}[\ell]$ and append result to $M_{a\alpha b}$
        end if
        $\ell \leftarrow \ell + 1$ ▷ Proceed to next element in $M_{\alpha b}$
      end while
      $i \leftarrow i + 1$ ▷ Proceed to next element in $M_{a\alpha}$
    end while
  end while
  return $M_{a\alpha b}$
end function

Result: Sorted list of corpus locations matching source language pattern $a\alpha b$: $M_{a\alpha b}$

Figure 2. Query intersection algorithm implemented in Joshua. This algorithm is adapted from a corrected version (Lopez, p.c.) of query intersection (Lopez, 2008).

cache is maintained. This cache maps from source phrase to the corresponding set of translation rules.

Another technique used to save memory is the option of using memory-mapped data structures. Memory-mapped version of the corpus array, suffix array, and alignment grids data structures are implemented and used by default.

4. Using Joshua

Given a word-aligned parallel corpus, the first step in extracting a grammar, either at run-time or during a preprocessing stage, is to compile the memory-mappable data structures to binary files on disk. The `joshua.corpus.suffix_array.Compile` program takes four parameters: source corpus text file, target corpus text file, word alignments text file, and an output directory path. The output directory, by convention, is named with the suffix `.josh`. This directory stores the binary representations of the symbol table, source and target corpus arrays, and the source and target suffix arrays. These binary files are given canonical names inside the `.josh` directory, so that the decoder can use them simply by specifying the `.josh` directory in the `tm_file=...` line of the Joshua configuration file.

It is often useful (especially during MERT) to extract a test set specific grammar once in a preprocessing step, since that test set will be translated many times and re-extracting the grammar each time would be wasteful. To perform this task, the program `joshua.prefix_tree.ExtractRules` can be used. When run directly, this program accepts either three arguments (a compiled `.josh` directory, file name for grammar to extract, and test file) or five arguments (source corpus text file, target corpus text file, word alignments text file, file name for grammar to extract, and test file). The following subsections document the mandatory and optional parameters that can be passed to this program through the `extractRules` ant task.

4.1. Mandatory parameters

testFile Path to plain text file containing a source language test file. The grammar extracted by `extractRules` will be all of the rules required to translate the sentences in this test file.

outputFile Path where extracted grammar will be placed. This grammar will consist of all of the rules required to translate the sentences in the test file defined in the `testFile` option.

joshDir Path to directory containing the binary files representing memory-mappable aligned parallel corpus.

The following parameters may be specified instead of the `joshDir` parameter.

sourceFileName Path to text file containing source corpus.

targetFileName Path to text file containing target corpus.

alignmentsFileName Path to text file containing word alignment data.

4.2. Optional parameters

maxPhraseSpan Defines the maximum span (in the source corpus) of any extracted SCFG rule. Default value is `10`.

maxPhraseLength Defines the maximum number of tokens (terminals plus nonterminals) allowed in the source right-hand side of any extracted SCFG rule. Default value is `10`.

maxNonterminals Defines the maximum number of nonterminal symbols allowed in the source side of any synchronous context-free rule extracted. Note: the number and type of nonterminals is the same in the source and target right-hand sides of a SCFG rule. Default value is `2`.

cacheSize Maximum number of source phrases for which translation rules will be maintained in the least-recently-used (LRU) cache.

encoding Defines the file encoding scheme of the input test file and the output grammar file. Default value is `UTF-8`.

ruleSampleSize When extracting SCFG rules for a given source language phrase, this option defines the number of instances of that source phrase will be sampled from the source training corpus for use in rule extraction. Default value is `300`.

startingSentence Defines the (1-based) sentence index in the test file where grammar extraction will begin. Default value is 1, indicating that a grammar will be extracted capable of translating sentences starting with the first sentence in the test file.

maxTestSentences Defines the number of sentences in the test file over which grammar extraction will be performed. Default value is `Integer.MAX_VALUE`. For example, given a test file of 100 sentences, the options `startingSentence="51" maxTestSentences="25"` would cause grammar extraction to be performed over test sentences 51–75.

The following parameters can be configured in the `extractRules` target to make rule extraction behave like the Hiero suffix array rule extractor (Lopez, 2008) instead of the behaving according to the rule extraction originally defined in Chiang (2005).

sentenceInitialX Boolean option indicates whether rules with an initial source-side nonterminal should be extracted from phrases at the start of a sentence, even though such rules do not have supporting corporal evidence. This option is provided for compatibility with Hiero's suffix array rule extractor (Lopez, 2008), in which this setting is set to true. Default value is `true`.

sentenceFinalX Boolean option indicates whether rules with a final source-side nonterminal should be extracted from phrases at the end of a sentence, even though such rules do not have supporting corporal evidence. This option is provided for compatibility with Hiero's suffix array rule extractor (Lopez, 2008), in which this setting is set to true. Default value is `true`.

edgeXViolates Boolean option indicates whether rules with an initial or final source-side nonterminal should be extracted when the source corpus phrase span for

the rule, discounting the initial or final nonterminal, is already equal to the maximum phrase span value. Since nonterminals conceptually correspond to elided elements in the training corpus, setting this value to true allows phrases which have a longer phrase span than the maximum allowed phrase span. This option is provided for compatibility with Hiero's suffix array rule extractor (Lopez, 2008), in which this setting is set to true. Default value is `true`.

requireTightSpans Boolean option; if true, follow the heuristic from (Chiang, 2005): where multiple initial phrase pairs contain the same set of alignment points, consider only the smallest when performing rule extraction. For compatibility with Lopez (2008), set this parameter to false. Default value is `true`.

Additional options can be configured in the `extractRules` target to change the behavior of the prefix tree.

keepTree Boolean option indicates whether the prefix tree should persist from sentence to sentence during grammar extraction. If set to false, a new prefix tree will be created to process each sentence in the test file. Default value is `true`.

printPrefixTree Boolean option indicates whether a representation of the prefix tree should be printed to standard output. If set to true, the tree will be printed after processing each sentence in the test file. Default value is `false`.

Acknowledgements

This research was supported in part by the EuroMatrixPlus project funded by the European Commission under the Seventh Framework Programme, by the US National Science Foundation under grant IIS-0713448, and by the DARPA GALE program under contract number HR0011-06-2-0001. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

Special thanks to Adam Lopez for his help and advice, and for making the LaTeXcode for his algorithms available.

Bibliography

Brown, Ralf D. A modified burrows-wheeler transform for highly-scalable example-based translation. In *Proceedings of the 6th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2004)*, Washington DC, 2004.

Callison-Burch, Chris, Colin Bannard, and Josh Schroeder. 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, 2005.

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

Fredkin, Edward. Trie memory. In *Communications of the ACM*, volume 3, pages 490–499, 1960.

Koehn, Philipp. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the 6th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2004)*, Washington DC, 2004. URL http://www.iccs.informatics.ed.ac.uk/~pkoehn/publications/pharaoh-amta2004.pdf.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference of the North American chapter of the Association for Computational Linguistics (HLT/NAACL-2003)*, Edmonton, Alberta, 2003. URL http://www.isi.edu/~koehn/publications/phrase2003.pdf.

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. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL-2007 Demo and Poster Sessions*, Prague, Czech Republic, 2007.

Kumar, Shankar, Yonggang Deng, and William Byrne. A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12 (1):35–75, 2006.

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. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W09/W09-0x24.

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

Lopez, Adam. *Machine Translation by Pattern Matching*. PhD thesis, University of Maryland, March 2008.

Manber, Udi and Gene Myers. Suffix arrays: A new method for on-line string searches. In *The First Annual ACM-SIAM Symposium on Dicrete Algorithms*, pages 319–327, 1990.

Marcu, Daniel and William Wong. A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-2002)*, Philadelphia, Pennsylvania, 2002. URL http://www.isi.edu/~marcu/papers/jointmt2002.pdf.

Nagao, Makoto. A framework of a mechanical translation between Japanese and English by analogy principle. In Elithorn, A. and R. Banerji, editors, *Artificial and Human Intelligence: edited review papers presented at the international NATO Symposium*, pages 173–180. 1981.

Och, Franz Josef and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449, 2004. URL http://acl.ldc.upenn.edu/J/J04/J04-4002.pdf.

Och, Franz Josef, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proceedings of the Joint Conference of Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999. URL http://www.isi.edu/~koehn/publications/phrase2003.pdf.

Phillips, Aaron B. and Ralf D. Brown. Cunei machine translation platform: System description. In *3rd Workshop on Example-Based Machine Translation*, Dublin, Ireland, 2009.

Sato, Satoshi and Makoto Nagao. Toward memory-based translation. In *Papers presented to the 13th International Conference on Computational Linguistics (CoLing-1990)*. 1990. URL http://acl.ldc.upenn.edu/C/C90/C90-3044.pdf.

Somers, Harold. An overview of EBMT. In Carl, Michael and Andy Way, editors, *Recent Advances in Example-Based Machine Translation*, chapter 4, pages 115–153. Kluwer Academic Publishers, 2003.

Zhang, Ying and Stephan Vogel. An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proceedings of the 10th Annual Conference of the European Association for Machine Translation (EAMT-2005)*, Budapest, Hungary, 2005.

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.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010

INSTRUCTIONS FOR AUTHORS

Manuscripts are welcome provided that they have not yet been published elsewhere and that they bring some interesting and new insights contributing to the broad field of computational linguistics in any of its aspects, or of linguistic theory. The submitted articles may be:

- long articles with completed, wide-impact research results both theoretical and practical, and/or new formalisms for linguistic analysis and their implementation and application on linguistic data sets, or
- short or long articles that are abstracts or extracts of Master's and PhD thesis, with the most interesting and/or promising results described. Also
- short or long articles looking forward that base their views on proper and deep analysis of the current situation in various subjects within the field are invited, as well as
- short articles about current advanced research of both theoretical and applied nature, with very specific (and perhaps narrow, but well-defined) target goal in all areas of language and speech processing, to give the opportunity to junior researchers to publish as soon as possible;
- short articles that contain contraversing, polemic or otherwise unusual views, supported but some experimental evidence but not necessarily evaluated in the usual sense are also welcome.

The recommended length of long article is 12–30 pages and of short paper is 6-15 pages.

The copyright of papers accepted for publication remains with the author. The editors reserve the right to make editorial revisions but these revisions and changes have to be approved by the author(s). Book reviews and short book notices are also appreciated.

The manuscripts are reviewed by 2 independent reviewers, at least one of them being a member of the international Editorial Board.

Authors receive two copies of the relevant issue of the PBML together with 10 offprints of their article.

The guidelines for the technical shape of the contributions are found on the web site `http://ufal.mff.cuni.cz/pbml.html`. If there are any technical problems, please contact the editorial staff at `pbml@ufal.mff.cuni.cz`.