The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009

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 978-80-904175-1-9
ISSN 0032-6585

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 91 JANUARY 2009

CONTENTS

Editorial 5

Articles

Improved Minimum Error Rate Training in Moses 7
Nicola Bertoldi, Barry Haddow, Jean-Baptiste Fouet

Memory-Based Machine Translation and Language Modeling 17
Antal van den Bosch, Peter Berck

PostCAT – Posterior Constrained Alignment Toolkit 27
João Graça, Kuzman Ganchev, Ben Taskar

An Open Source Rule Induction Tool for Transfer-Based SMT 37
Yvette Graham, Josef van Genabith

Decoding in Joshua 47
Open Source, Parsing-Based Machine Translation
Zhifei Li, Chris Callison-Burch, Sanjeev Khudanpur, Wren Thornton

apertium-cy – a collaboratively-developed free RBMT system for Welsh to English 57
Francis Tyers, Kevin Donnelly

Grammar based statistical MT on Hadoop 67
An end-to-end toolkit for large scale PSCFG based MT
Ashish Venugopal, Andreas Zollmann

Z-MERT: A Fully Configurable Open Source Tool for Minimum Error Rate Training of Machine Translation Systems 79
Omar F. Zaidan

Unsupervised Generation of Parallel Treebanks through Sub-Tree Alignment 89
Ventsislav Zhechev

Instructions for Authors 99

List of Authors 101

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009

EDITORIAL

Special Issue on Open Source Machine Translation Tools

We live in exciting times for machine translation. The field is making rapid progress due to the refinement of statistical methods and the convergence of statistical, rule-based, and knowledge-based approaches and increased linguistic sophistication of the employed models. Machine translation has found its way to everyday use through web based services such as by Systran and Google. An increasing number of research papers are published by a growing research community.

While the growing complexity and refinement of methods leads to advances in the field, there is also the danger that it becomes too hard for a newcomer to build a machine translation system in her garage (or, more commonly, her graduate research office): there are just too many tools to be build and methods to be mastered. We believe that it is essential for maintaining machine translation as a vivid academic research field that tools and resources are most widely shared. The history of the field so far has shown that advances come from many different directions, often from newcomers entering the field. This is possible due to a environment of shared tools and resources.

The goal of a serious of annual workshops called "Machine Translation Marathon" that started in 2007 under the EU-funded EuroMatrix project (Framework Programme 6) has been to disseminate methods and tools for machine translation. This year, the Third Machine Translation Marathon, held January 26–30 in Prague, also hosts a open source convention. We hope to encourage the sharing of open source resources by providing a such a forum.

We solicited papers that describe open source tools for machine translation. We selected among the submissions nine papers that are assembled in this special issue of the Prague Bulletin of Mathematical Linguistics (PBML No. 91). The papers cover different approaches to machine translation — ranging from rule-based to statistical — and describe full systems and specialized methods. All the papers describe tools that are readily available, and thus enable future research to start novel research in the field without spending too much time catching up with the state of the art.

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

Note:
The index to the volumes 81–90 (2004–2008) will be published in the next regular issue of the PBML No. 92 (2009).

Eva Hajičová
Editor-in-Chief
hajicova@ufal.mff.cuni.cz

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 7–16

Improved Minimum Error Rate Training in Moses

Nicola Bertoldi, Barry Haddow, Jean-Baptiste Fouet

Abstract

We describe an open-source software for minimum error rate training (MERT) for statistical machine translation (SMT). This was implemented within the Moses toolkit, although it is essentially standsalone, with the aim of replacing the existing implementation with a cleaner, more flexible design, in order to facilitate further research in weight optimisation. A description of the design is given, as well as experiments to compare performance with the previous implementation and to demonstrate extensibility.

1. Introduction

1.1. Background

In Statistical Machine Translation (SMT), probabilistic models are used to find the best possible target translation $\mathbf{e}^*$ of a given source sentence $\mathbf{f}$, amongst all possible translations $\mathbf{e}$. The search for the best translation is known as *decoding*. The probabilistic models are estimated from bilingual and monolingual training data, and may include translation models, language models, reordering models, etc. In order to combine evidence from different models, it is standard practice to use a discriminative linear model, with the log probabilities as features. If the features of the model are $h_1, \ldots, h_r$, which depend on $\mathbf{e}$ and $\mathbf{f}$, then the best translation is given by

$$\mathbf{e}^*(\lambda) = \arg\max_{\mathbf{e}} \sum_{i=1}^{r} \lambda_i h_i(\mathbf{e}, \mathbf{f})$$

and depends on the feature weights $\lambda_1, \ldots, \lambda_r$.

The advantage of combining log probabilities in such a linear model is that the features can be arbitrary functions of the source and target sentences, and are not limited to just being log probabilities. For instance, a word count feature can be added which will penalize long or short sentences depending on the sign of the corresponding weight.

Please cite this article as: Nicola Bertoldi, Barry Haddow, Jean-Baptiste Fouet, Improved Minimum Error Rate Training in Moses. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 7–16.

The problem then arises of how to optimize the feature weights, in other words how to find a set of weights which will offer the best translation quality. The standard solution is to use minimum error rate training (MERT), a procedure introduced by Och (2003), which searches for weights minimizing a given error measure, or, equivalently, maximizing a given translation metric. This algorithm enables the weights to be optimized so that the decoder produces the best translations (according to some automatic metric Err and one or more references ref) on a development set of parallel sentences.

$$\lambda^* = \arg\min_{\lambda} \mathbf{Err}(\mathbf{e}^*(\lambda); \mathbf{ref})$$

The main feature of Och's approach is the exploitation of n-best translation alternatives (for each input sentence), that allows for fast convergence of the optimization process. The translation metric most commonly employed as the objective in MERT is the BLEU score (Papineni et al., 2001), although any automatic metric could in principle be used.

The weight optimization algorithm introduced in Och (2003) (and more fully described in Koehn (forthcoming)) is a form of coordinate ascent, where the search updates the feature weight which appears most likely to offer improvements, then iterates. Since calculation of the objective (in other words, the translation metric) is quite expensive, as much of it as possible is pre-calculated before running the optimization. Because the error surface is highly non-convex, MERT is always at risk of being trapped at local maxima; and because it uses n-best lists as an approximation for the decoder output, it cannot explore the actual parameter space. However, despite its limitations, MERT tends to produce good results.

MERT is the subject of ongoing research, for example to reduce the local maxima problem using regularization and stochastic search (Cer et al., 2008), to make convergence faster and more robust by selecting starting points by random walks (Moore and Quirk, 2008), and to replace the n-best lists by lattices (Macherey et al., 2008) and thereby improve the estimates of the expected translation score.

The MERT implementation discussed in this article is a subproject of Moses (Koehn et al., 2007), one of the leading open source implementations of phrase-based machine translation (Koehn et al., 2003). Having a state-of-the art SMT system available as open source has been proved to be a successful way of enabling and stimulating research in the area, as researchers do not have to invest large amounts of effort in reimplementing the work of others. In order to improve on the current best systems, they can take Moses as starting point, learn from its open code, and implement their own proposed improvements on top of it.

1.2. Motivations for New Software

As stated in the previous Section, MERT is a crucial step for optimally tuning any SMT system based on a discriminative linear model. Consequently, any possible enhancement of MERT could improve the overall performance of an SMT system.

Moreover, an effective and efficient implementation of MERT is fundamental per se, and essentially independent from the MT engine. In fact, weight optimization is still an open issue

which the MT community continues researching on, especially in regard to convergence issues. The availability of a flexible and modular open source software could help this research.

Finally, the original implementation of MERT provided with Moses is a collection of scripts written in several programming language and in different periods. The interaction of the modules is not optimal, and consequently its efficiency is quite limited. The old version of MERT strongly relies on BLEU (Papineni et al., 2001) as an automatic MT measure, and on the weight optimization criterion proposed by (Och, 2003); adding new automatic MT measures or new optimization algorithms would have been hard.

For these reasons, during the Second MT Marathon[1] held in Berlin in 2008, it was decided to implement MERT in a new standalone open-source software and to isolate it from Moses as much as possible. The new implementation was defined from scratch aiming at i) improving efficiency in terms of computation time, runtime memory consumption, storage disk space, etc.; ii) increasing modularity and flexibility to easily allow for new scoring measures and new optimization criteria; iii) parallelizing some steps of the optimization process. The core of the software was written in C++. The new MERT software is licensed under the LGPL[2].

A detail documentation how to use the software can be found online[3].

2. Design and Implementation

2.1. System Outline

Using a small (typically 500-1000) set of parallel sentences (the *tuning set*) MERT attempts to find a set of feature weights which maximize the decoder performance on this set. The full MERT algorithm consists of an outer loop and an inner loop (as illustrated in Figure 1). The outer loop runs the decoder over the source sentences in the tuning set with a given set of feature weights, generating n-best lists of translations, and then calls the inner loop to optimize the weights based on those n-best lists, repeating until the weights no longer change. In the inner loop, an iterative line optimization algorithm (Och, 2003, Koehn, forthcoming) is applied to search for the highest scoring feature weights using estimates of the decoder score derived using the n-best lists.

To ensure that the n-best lists are as diverse as possible, the n-best lists produced by each run of the decoder are merged with those produced by the previous runs. The number of previous n-best lists can be chosen at runtime.

Within Moses, the outer loop was implemented using a perl script (`mert-moses.pl`) and the original implementation of the inner loop (`score-nbest.py` and `cmert`) consisted of perl, python and C programs. The main focus of the improved MERT was the inner loop, which was completely rewritten from scratch, although a new version of the outer loop script (`mert-moses-new.pl`) was also created as it was necessary to change the interface between the inner and outer loops.

[1] `http://www.statmt.org/mtm2/`

[2] `http://www.gnu.org/licenses/lgpl.html`

[3] http://www.statmt.org/moses/?n=FactoredTraining.Tuning

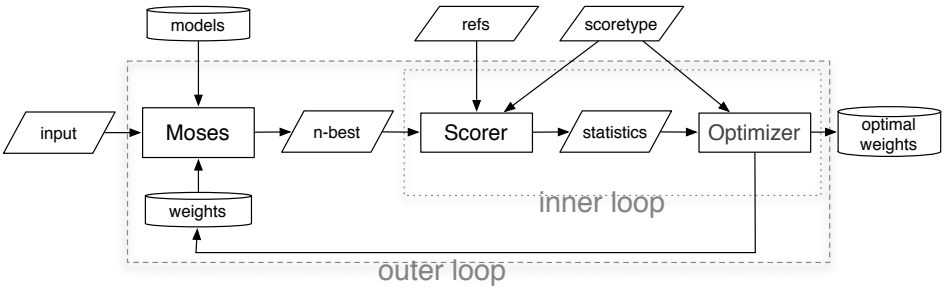

Figure 1. Outer and inner loops of MERT

Conceptually the inner loop consists of two components — the scorer and the optimizer — which are made explicit by the object-oriented design of the new MERT. The job of the scorer is to use an automatic metric (e.g. BLEU, PER, METEOR) to score a given ranking of the n-best lists, whilst the optimizer performs the actual parameter optimization. For efficiency purposes, as much of the metric calculation as possible is performed prior to running the optimization, because the optimizer will have to score a large number of translation hypothesis. For the BLEU scorer, for example, all the n-gram statistics are precalculated. The new MERT implementation is currently split into two processes; `extractor`, which does the scoring precalculations, and `mert`, which does the optimization. The `extractor` also has the job of extracting the feature values corresponding to each hypothesis in the n-best list, and making them available to the optimizer `mert`.

There is also a regression testing framework for the new MERT, which simulates the outer loop with pre-prepared data and makes it possible to check that the optimized weights agrees with the expected. The testing framework produces timing information to enable the monitoring of MERT runtime performance.

2.2. Object Model

In Figure 2 the main classes in the new MERT implementation are shown, using UML[4]. The `Scorer` class is the abstract base class of all scorers, and currently has two concrete subclasses; `BleuScorer`, which implements BLEU, and `PerScorer`, which implements position-independent recognition rate. The main work of the `Scorer` is done in the two statistics precalculation methods (`setReferenceFiles()` and `prepareStats()`) and the main scoring method (`score()`), which is used by the optimizer. In the scoring class hierarchy there is also another abstract class (`StatisticsBasedScorer`). This is used to abstract out common features of automatic translation metrics that are calculated by adding some statistics across all examples in the test set and then performing a calculation on the totals.

The optimization strategy is also encapsulated in a class, the `Optimizer`, which currently has one concrete subclass which implements the line optimization algorithm

[4]Unified Modelling Language - see `www.uml.org` for more information.

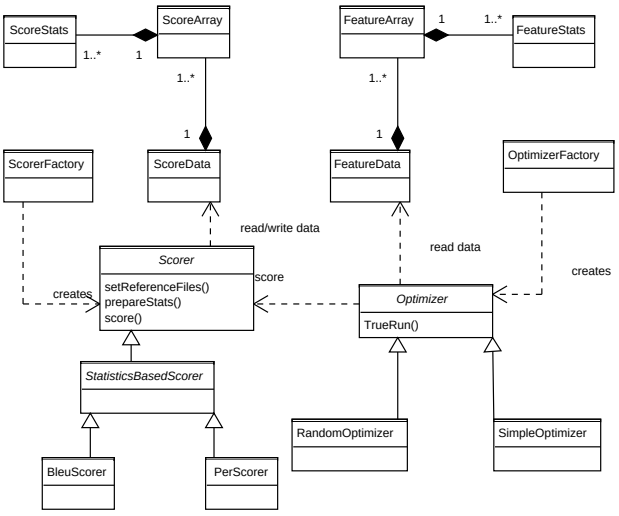

Figure 2. Class diagram for the MERT implementation

(`SimpleOptimizer`) and another which merely performs a baseline random optimization. Another optimization strategy could easily be added by subclassing `Optimizer` and overriding the `TrueRun()` method. Both the scoring strategy classes and the optimizer strategy classes have corresponding factories which are used to construct an instance of the appropriate type.

In addition to scoring and optimization, the other important set of classes are those that concern input/output, and these are the `Score*` and `Feature*` classes found at the top of Figure 2. Recall that at the start of the inner loop, the `extractor` processes the `n`-best lists, extracting scoring statistics and feature values, and passes them to the `mert` process. The file system is used to interface between these two processes, and the input/output classes are used to load and save the data in either textual or binary format. The former is easier to debug, whilst the latter offers better performance. `extractor` also transforms data from/to textual and binary formats.

3. Evaluation

In this Section we compare the old and new implementations of MERT. First, we check that they are similarly effective, i.e. that they provide similar optimized weights, and consequently achieve similar translation performance. Then, we measure the efficiency of the two versions in terms of disk occupancy and computation time. Finally, we present an add-on of MERT confirming the ease of extensibility of the new implementation.

3.1. Translation Performance

In order to verify that the new MERT works correctly, it was tested on two French-English translation tasks using data provided for the Third Workshop on Statistical Machine Transla-

tion (WMT08) (Callison-Burch et al., 2008), and the results compared to those achieved by the old MERT implementation. In the first experiment, Moses was trained on the 2007 release of the news-commentary (nc) parallel training set, using the target side of this for language modelling. The nc-dev07 set was used for tuning on 100-best lists and the decoder was tested on nc-devtest07, nc-test07 and newstest08. The BLEU scores for the old and new MERT implementations are shown in Table 1.

	nc-devtest07	nc-test07	newstest08
old mert	24.42	25.55	15.50
new mert	24.87	25.70	15.54

*Table 1. Comparison of performance (*bleu*) of old and new MERT implementations, using the news commentary training and test data.*

The second experiment also used French-English data from the WMT08 workshop, but this time Moses was trained, tuned and tested on europarl extracts. The training data was the europarl v3 release, the tuning set was dev06 and the test sets were devtest06, test06 and test07. Again both the new and old MERT implementations were used for tuning with 100-best lists. The BLEU scores are shown in Table 2. From the results of these two experiments, it can be

	devtest06	test06	test07
old mert	32.75	32.67	33.23
new mert	32.86	32.79	33.19

*Table 2. Comparison of performance (*bleu*) of old and new MERT implementations, using the europarl training and test data.*

seen that the weights learnt by the old MERT and new MERT lead to translation performances which are virtually indistinguishable.

3.2. Space and Time Performance

We also compared the efficiency of the old and new MERT implementations in terms of disk occupancy and computation time, we ran the two versions on a development data (dev06) of the Spanish-English translation tasks of WMT08 workshop. Moses was trained in a standard way on the data provided for this task. At each iteration we generated 200-best alternatives for each of the 2000 input sentences in the tuning set. Extraction of score statistics and weight optimization were performed on one 64bit Intel Xeon CPU 3.20GHz machine.

The number of n-best lists used by the new MERT software to optimize weights, is configurable at runtime: in addition to the last generated list, it can exploit from 0 to all previous ones. We ran the new MERT software in three different conditions, namely using 1, 3, and all previous n-best lists. Each iteration of the optimization process of the four MERT configurations, namely `old`, `new-all`, `new-3`, `new-1`, produces a set of weights, and hence a specific system. These systems are evaluated both on the dev06 and the test08; BLEU scores are shown in Figure 3, respectively.

For some reason to investigate more deeply, the second iteration of the new MERT produces very bad weights, which gives performance close to 0. In any case, this behavior was already observed by Macherey et al. (2008), who attribute the performance drop to an overfitting issue.

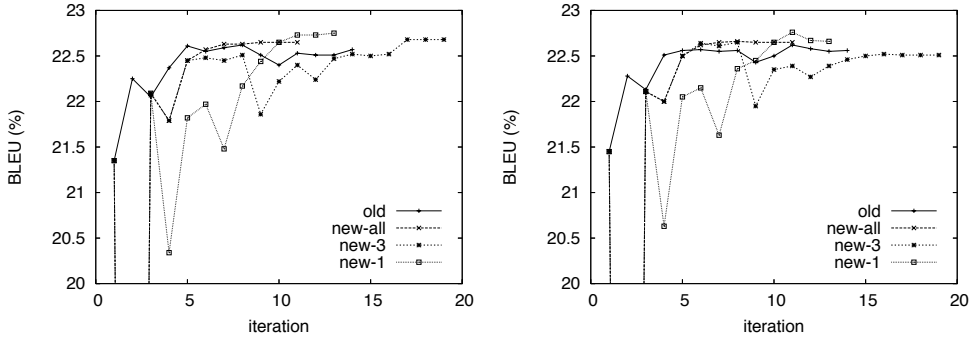

Figure 3. bleu *score of different versions of MERT on dev06 (left) and test08 (right).*

Both MERT implementations converges after 5/6 iterations, when using all the previous n-best lists; while convergence is slower and less stable if new MERT uses fewer.

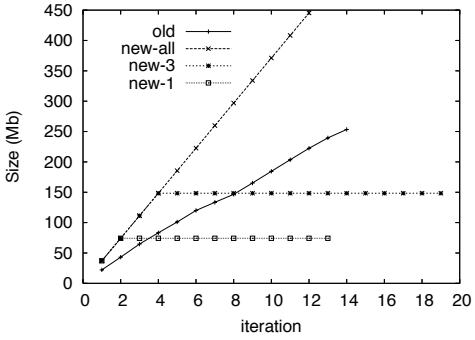

Figure 4. Global size (in Mbytes) of the files needed to the MERT implementations at each iterations.

Figure 4 reports the total size (in Mbytes) of files needed to store all the required statistics for the weight optimization at a given iteration. Figures are given for compressed[5] files for the old versions and for the binary files of the new one. New implementation requires twice a bigger disk occupancy; but limiting the number of previous n-best lists taken in consideration for optimization allows to maintain the disk usage constant.

The plot on the left of Figure 5 shows time (in seconds) to perform a single iteration, excluding time for decoding and generating the actual n-best list because independent from the

[5]Compression is performed by the gzip command of Linux with its default compression level (6).

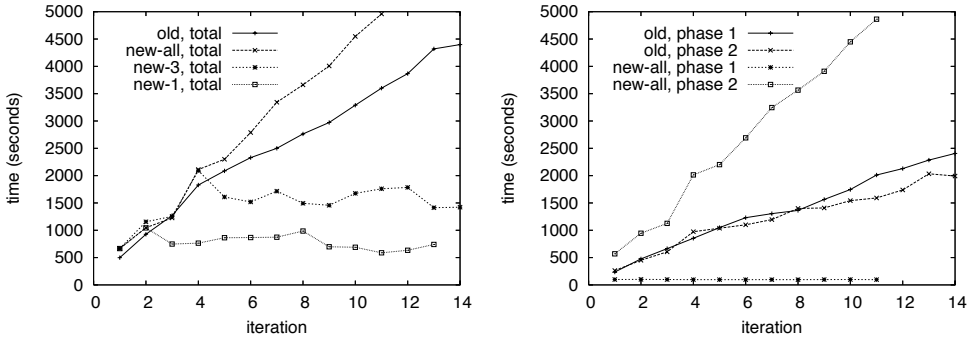

Figure 5. Time for performing a single iteration (left). Time for extracting features and computing score statistics (phase 1), and for optimizing weights (phase 2) (right).

inner loop of any MERT implementations. The plot on the right reports separately the time of the two phases which the inner loop is divided in: 1) extracting feature scores, computing score statistics and saving on the disk, and 2) optimizing weights.

In phase 1 the old MERT implementation sorts actual translation candidates and removes duplicates of those already observed in previous iterations; instead, the new one computes and stores information for all candidates of the actual n-best list. Consequently, in phase 2 the old version searches the best weights over a smaller set of candidates than the new one. Hence, the former takes a time proportional to the stored candidates in both phases; while the latter requires a constant time for the first phase and a larger time for the second one. The gap slightly expands as the number of iterations (and candidates) increases. Again, the new MERT implementation easily allows to bound computation time by taking into account fewer iterations.

It is worth noticing that new MERT software is still under development, and the removal of duplicates has the highest priority in the agenda of enhancements.

3.3. Extensibility

Since one of the aims of the re-implementation of MERT was to provide a cleaner design and so improve the extensibility of the codebase, a useful opportunity to test this extensibility was presented by some recently published improvements to the MERT algorithm. It was shown by Cer et al. (2008) that by using a form of "regularization", the risk of MERT being misled by spurious maxima (i.e. spikes in the error surface) could be reduced, leading to improvements in translation performance. The central idea is that, when performing a line search for the best BLEU score, instead of taking the BLEU scores at each point, the BLEU scores are "smoothed out" across a neighborhood of the point. This smoothing out is accomplished by one of two strategies; taking the minimum or taking the average.

When implementing the regularization method (Cer et al., 2008), it was clear that it was not just applicable to BLEU, and so was implemented higher up the class hierarchy (in `Statis-`

`ticsBasedScorer` - see Figure 2) so that it could be used with other scoring schemes. The implementation was a straightforward addition to the `score()` method.

To test the effect of the regularization on translation performance, experiments were performed using the WMT08 data for the language pairs French-English and German-English. The europarl parallel training set was used for training, with the europarl monolingual for language modelling, dev06 for tuning, and the resulting system tested on devtest06, test06 and test07. The translation scores are shown in Table 3. Unfortunately no clear pattern is visible in

		en-fr			en-de		
Method	Window	devtest06	test06	test07	devtest06	test06	test07
none	n/a	32.86	32.79	33.19	27.54	27.67	28.07
minimum	±1	32.70	32.65	33.20	27.51	27.79	28.00
	±2	32.81	32.75	33.21	27.75	27.85	28.10
	±3	32.83	32.76	32.93	27.70	27.92	27.96
	±4	32.88	32.77	33.24	27.70	27.87	28.02
average	±1	32.79	32.77	33.29	27.44	27.81	28.00
	±2	32.89	32.83	33.28	27.63	27.73	27.98
	±3	32.78	32.67	33.19	27.53	27.67	27.87
	±4	32.81	32.79	33.25	27.81	28.01	28.22

Table 3. Experiments with mert regularization

the results in Table 3. Examination of the error surfaces explored by MERT suggests that they are fairly smooth and not prone to the sort of sudden spikes that this regularisation is designed to smooth out. It is hypothesised that for unrelated language pairs (such as Chinese-English) these kind of irregularities in the error surface are more common than for related (and therefore easier) language pairs such as German-English and French-English. Further investigation would be required to confirm this hypothesis.

Developers have already added Position Independent Error Rate (PER) as an alternative automatic translation score. Results are not reported, because they essentially confirm those achieved by exploiting BLEU.

4. Conclusion and Future Work

In this paper we presented a new open-source software implementing MERT. Although it is still distributed under Moses toolkit, it is essentially a standalone piece of software. The most important characteristic of the distribution is its modularity which allows an easy extensibility to new error measures and enhanced optimization algorithms. At the moment, BLEU score and PER are implemented and two optimization criteria can be chosen: a (possibly smoothed) line-wise optimization, and a dummy random optimization.

New version of MERT actually requires more disk usage and is slightly slower than the previous one, because it does not remove duplicate translations from the n-best lists. This issue will be addressed very soon. The parallelization of portions of the algorithm is also in the close-term agenda, to reduce computational time.

In the close future, developers will also work on i) adding new automatic measures, like WER and NIST score, and combination of them, ii) constraining the space of the feature weights, iii) adding priors to weights and iv) implementing the lattice optimization as proposed by Macherey et al. (2008).

Bibliography

Callison-Burch, Chris, Philipp Koehn, Christof Monz, Josh Schroeder, and Cameron Shaw Fordyce, editors. 2008. *Proceedings of the Third Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Columbus, Ohio.

Cer, Daniel, Dan Jurafsky, and Christopher D. Manning. 2008. Regularization and search for minimum error rate training. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 26–34, Columbus, Ohio.

Koehn, Philipp. forthcoming. *Statistical Machine Translation*. Cambridge University Press.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demo Session*, pages 177–180, Prague, Czech Republic.

Koehn, Philipp, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT-NAACL*, pages 127–133, Edmonton, Canada.

Macherey, Wolfgang, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. 2008. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of EMNLP*, pages 725–734, Honolulu, Hawaii.

Moore, Robert C. and Chris Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of Coling*, pages 585–592, Manchester, UK.

Och, Franz Josef. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167, Morristown, NJ, USA.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. Bleu: a method for automatic evaluation of machine translation. Research Report RC22176, IBM Research Division, Thomas J. Watson Research Center.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 17–26

Memory-Based Machine Translation and Language Modeling

Antal van den Bosch, Peter Berck

Abstract

We describe a freely available open source memory-based machine translation system, MBMT. Its translation model is a fast approximate memory-based classifier, trained to map trigrams of source-language words onto trigrams of target-language words. In a second decoding step, the predicted trigrams are rearranged according to their overlap, and candidate output sequences are ranked according to a memory-based language model. We report on the scaling abilities of the memory-based approach, observing fast training and testing times, and linear scaling behavior in speed and memory costs. The system is released as an open source software package[1], for which we provide a first reference guide.

1. Introduction

Recently, several independent proposals have been formulated to integrate discrete classifiers in phrase-based statistical machine translation, to filter the generation of output phrases (Bangalore, Haffner, and Kanthak, 2007, Carpuat and Wu, 2007, Giménez and Màrquez, 2007, Stroppa, Van den Bosch, and Way, 2007), all reporting positive effects. This development appears an interesting step in the further development of statistical machine translation. These same developments can also be employed to produce simple but efficient stand-alone translation models.

In this paper, we introduce MBMT, memory-based machine translation. The memory-based approach, based on the idea that new instances of a task can be solved by analogy to similar instances of the task seen earlier in training and stored in memory as such, has been used successfully before in various NLP areas; for an overview, see (Daelemans and Van den Bosch, 2005).

MBMT is a stand-alone translation model with a simple decoder on top that relies on a memory-based language model. With a statistical word alignment as the starting point, such

[1] http://ilk.uvt.nl/mbmt

Please cite this article as: Antal van den Bosch, Peter Berck, Memory-Based Machine Translation and Language Modeling. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 17–26.

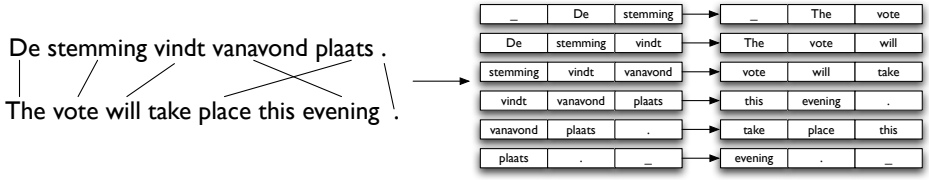

Figure 1. An example training pair of sentences, converted into six overlapping trigrams with their aligned trigram translations.

as produced by GIZA++ (Och and Ney, 2003), MBMT is shown to be very fast both in training and translation.

The overall architecture of the system is described in Section 2. A brief evaluation and reference guide of the system is provided in Section 3. The language modeling module, also made available as a separate language modeling toolkit, is described in Section 4. We wrap up in Section 5.

2. Memory-based machine translation

Memory-based machine translation (Van den Bosch, Stroppa, and Way, 2007) can be characterized as an instantiation of example-based machine translation (EBMT), as it essentially follows EBMT's basic steps (Carl and Way, 2003): given a sentence in the source language to be translated, it searches the source side of the corpus for close matches and their equivalent target language translations. Then, it identifies useful source–target fragments contained in those retrieved examples; and finally, it recombines relevant target language fragments to derive a translation of the input sentence.

The scope of the matching function implied in the first step is an important choice. We take a simplistic approach that assumes no linguistic knowledge; we use overlapping trigrams of words as the working units, both at the source language side and the target language side.

The process of translating a new sentence is divided into a local phase (corresponding to the first two steps in the EBMT process) in which memory-based translation of source trigrams to target trigrams takes place, and a global phase (corresponding to the third EBMT step) in which a translation of a sentence is assembled from the local predictions. We describe the two phases in the following two subsections.

2.1. Local classification

Both in training and in actual translation, when a new sentence in the source language is presented as input, it is first converted into windowed trigrams, where each token is taken as the center of a trigram once. The first trigram of the sentence contains an empty left element, and the last trigram contains an empty right element. At training time, each source language sentence is accompanied by a target language translation. We assume that word alignment has

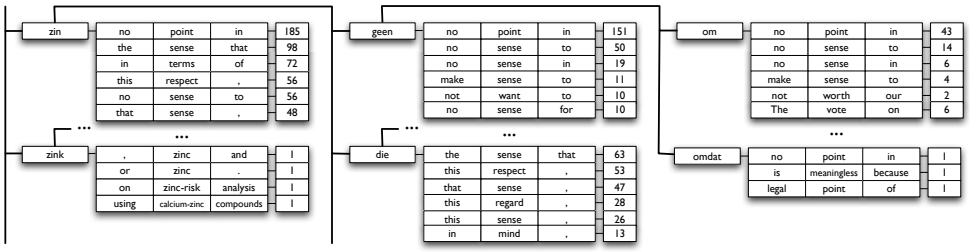

Figure 2. Excerpt of an mbmt igtree *structure, zooming in on the path represented by the input trigram "geen zin om", translatable to "no point in", among others.*

taken place, so that we know for each source word whether it maps to a target word, and if so, to which. Examples are only generated for source words that align to target words. Given the alignment, each source trigram is mapped to a target trigram of which the middle word is the target word to which the word in the middle of the source trigram aligns. The left and right neighboring words of the target trigram are the center word's actual neighbors in the target sentence. Figure 1 exemplifies the conversion of a training translation to six trigram mappings.

During translation, source trigrams are matched against the training set of stored source trigrams with a known mapping to a target trigram. The matching is carried out as a discrete classification. To this purpose we make use of IGTREE[2] (Daelemans, Van den Bosch, and Weijters, 1997), which compresses a database of labeled examples into a lossless-compression decision-tree structure that preserves the labeling information of all training examples. Figure 2 displays a fragment of the decision tree trained on the translation of Dutch trigrams to English. It highlights one path in the tree, representing the Dutch trigram "geen zin om" (translatable, among others, into "no point in" and "no sense to"). The tree encodes all possible trigram translations of, respectively, the middle word "zin", the bigram "geen zin", and the full trigram. This order reflects the information-gain weights of the three words with respect to predicting the output class.

During translation, IGTREE's classification algorithm traverses the decision tree, matching the middle, left, and right words of each new trigram to a path in the tree. Two outcomes are possible: (1) IGTREE finds a complete matching path, upon which it returns the most probable output trigram; (2) IGTREE fails to match a value along the way, upon which it returns the most probable output trigram given the matching path so far. Instead of the most probable path, it is also possible for IGTREE to return the full distribution of possible trigrams at the end of a matching path.

When translating new text, trigram outputs are generated for all words in each new source language sentence to be translated, since our system does not have clues as to which words would be aligned by statistical word alignment.

[2] `http://ilk.uvt.nl/timbl`

2.2. Global search

To convert the set of generated target trigrams into a full sentence translation, the overlap between the predicted trigrams is exploited. Figure 3 illustrates a perfect case of a resolution of the overlap (drawing on the example of Figure 1), causing words in the English sentence to change position with respect to their aligned Dutch counterparts. The first three English trigrams align one-to-one with the first three Dutch words. The fourth predicted English trigram, however, overlaps to its left with the fifth predicted trigram, in one position, and overlaps in two positions to the right with the sixth predicted trigram, suggesting that this part of the English sentence is positioned at the end. Note that in this example, the "fertility" words *take* and *this*, which are not aligned in the training trigram mappings (cf. Figure 1), play key roles in establishing trigram overlap.

In contrast to the ideal situation sketched in Figure 3, where one translation is produced, in practice many different candidate output sequences can be generated due to two reasons: first, each (potentially partially or fully incorrect) trigram may overlap with more than one trigram to the left or right, and second, the classifier may produce more than one output trigram at a single position, when it reaches a non-ending node with equally-probable trigram classes.

To select the most likely output among the potentially large pool of candidate outputs, we employ a memory-based target language model (Van den Bosch, 2006). This model, called WOPR, described in more detail in Section 4, is a word prediction IGTREE system trained on a monolingual target language corpus, which produces perplexity scores for each candidate output sequence presented to it.

WOPR provides the language model in a different way than most standard models do. Most models, like for example SRILM (Stolcke, 2002), estimate probabilities of words in context and build a back-off model containing n-grams to unigrams. WOPR uses a trigram model (it is not limited to trigrams, it could use any size and any context) but, because it uses the IGTREE algorithm, stores exceptions to default values rather than all n-grams. Other language models could be used, but we prefer to use WOPR because it uses the same IGTREE model also used as the core translation engine in the MBMT system. The model is described in more detail in Section 4.

As the number of possible output sequences may be large, MBMT currently applies Monte Carlo sampling to generate candidate output sequences to be scored by WOPR. This sampling is subject to a patience threshold p that halts the generation of new candidates when no improvement in perplexity scores is observed for p sample steps. By default, $p = 100$.

3. Evaluation and an Annotated Example

For the purpose of a brief evaluation, we first focus on the translation of Dutch to English, using the EMEA corpus, part of the Opus open source parallel corpus[3]. The EMEA corpus contains documents from the European Medicines Agency[4]. Texts in this corpus are of a re-

[3] `http://urd.let.rug.nl/tiedeman/OPUS/` – downloaded in June 2008.

[4] `http://www.emea.europa.eu/`

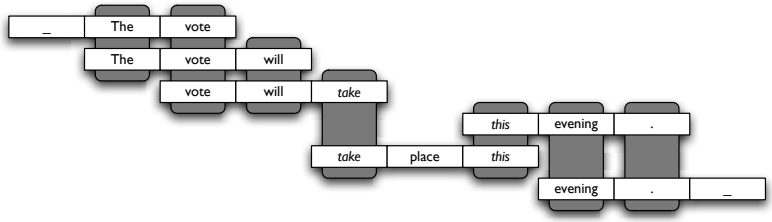

Figure 3. Producing a global solution by resolving the overlap between six trigrams. Italicized words are not aligned with source words.

stricted genre, consisting of quite formal, exact, and largely controlled language. We used the first 749,602 lines of text (approximately 9.0 million English and Dutch words). The corpus was split into a final 1,000-sentence test set and a training set containing the remainder of the data. The training sets were word-aligned using the GIZA++ algorithm (Och and Ney, 2003). No decapitalization was performed. The IGTREE-based language model used for translation is a single model trained on the first 112 million words of the Reuters RCV1 corpus.

We performed a learning curve experiment on the EMEA training set. We start at a training set size of 100,000 tokens, and increment with steps of 100,000 until 1 million tokens; then, we increment with steps of 1 million tokens up to the maximal training set size of 9 million tokens. The learning curve experiment serves to get an idea of the scaling abilities of MBMT in terms of performance; we also measure training and testing speeds and memory footprint. The learning curve experiment on the EMEA corpus produced performance curves of which we combine two in the left graph of Figure 4: the BLEU and METEOR (exact) scores. Both curves show a steady but somewhat weakening increase when the dataset doubles in size (note that the x axis is logarithmic).

Second, the middle graph of Figure 4 displays the number of seconds it takes to construct a decision tree, and to test. Testing occurs in a few seconds (up to eight seconds for 1,000 sentences, with an approximately linear increase of one second of testing time with each additional million training examples); the test graph virtually coincides with the x axis. Training times are more notable. The relation between training times and number of training examples appears to be linear; on average, each additional million of training examples makes training about 130 seconds slower.

Third, the right graph of Figure 4 shows a similar linear trend of the memory footprint needed by IGTREE and its decision tree, in terms of Megabytes. At 9 million training examples, the decision tree needs about 40 Mb, an average increase of 4.4 Mb per additional million examples.

As a second evaluation, we compare against the performance and training and testing times of MOSES on the EMEA corpus at the maximal training set size. Table 1 lists the performance on the test data according to word error rate, position-independent word error rate, BLEU, Meteor, and NIST. As is apparent from the results, MOSES performs at a markedly higher level

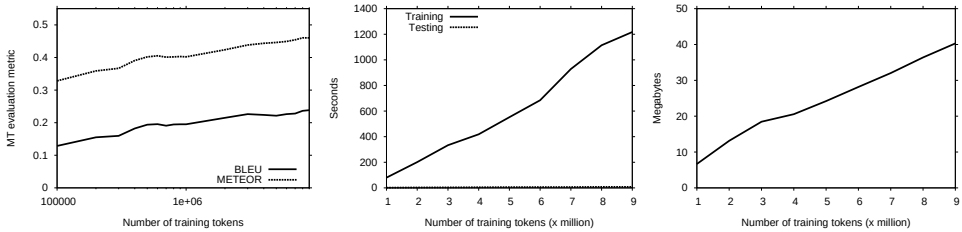

Figure 4. Learning curves on the emea *corpus in terms of MT evaluation (*bleu *and* meteor*, left, with a logarithmic x-axis), seconds to train and test (middle; the test graph virtually coincides with the x axis), and memory needed (right), with increasing amounts of training material available.*

of performance, but does so at the cost of a longer translation time: MBMT is about 20 times as fast. Training MBMT is about 10 times as fast as training MOSES; in both cases, the GIZA++ process has already been performed and is not included here.

System	WER	PER	BLEU	Meteor	NIST	Training (h:m:s)	Test (m:s)
MBMT	72.7	63.6	0.238	0.460	4.97	20:17	0:08
MOSES	46.6	39.4	0.470	0.650	7.06	3:10:06	2:51

Table 1. Comparing mbmt *agains* moses *on the* emea *corpus, in terms of five MT evaluation metrics, and training and testing times (elapsed wallclock time).*

Annotated Example

The MBMT software assumes a GIZA++-style A3 file, i.e. a word alignment file containing all aligned source and target training sentences, as training material. The software will convert this training file into an IGTREE decision tree, and is then capable of translating a raw text file in the source language (tokenized, one sentence per line) into a translated raw text file in the target language (also one sentence per line). The commandline functionality is currently limited to the identification of the A3 training file, and the source-language test text, plus the optional setting of the patience threshold to a non-default setting, e.g. $p = 50$, with the `-p` switch:

```
mbmt -t EMEA.9m.train.A3.final -t EMEA-dutch.test.txt -p50
```

During runtime, MBMT generates several intermediary files. First, the A3 file is converted to a training file suited for IGTREE, mapping source-language trigrams to aligning target-language trigrams (cf. Figure 1). Subsequently, this file is compressed into an IGTREE decision tree, at

a typical compression rate of about 95%. The test set is also converted into trigram instances (one instance per word), which are then classified by IGTREE. This output is stored in a file of which the first line looks as follows:

```
na de behandeling ? after_the_end { after_the_end 3.00000,
,_the_rate 3.00000, after_the_treatment 3.00000 }
```

The Dutch trigram *na de behandeling* ("after the treatment") is classified by IGTREE as mapping to three equally likely trigram translations. These three translations will be carried along to the final phase, where all predicted trigrams are used to generate possible translations, using a Monte Carlo sampling method with a halting criterion governed by the patience parameter p. Each candidate output sequence is scored by WOPR, the memory-based language model.

4. Memory-based language modeling

The MBMT system generates a candidate number of translations for each input sentence. The typical approximate solution to picking the best translation is to use a language model to determine which translation fits best in the target language, e.g. selecting the candidate string with the lowest perplexity score.

In the MBMT system, WOPR is the language model. It is a *word predictor* based on IGTREE, trained to predict the next word in a sentence (Van den Bosch, 2006). To calculate the perplexity of a sentence, we feed it to WOPR and see which words it predicts for each word in the sentence. The perplexity is calculated from the estimated probabilities of each prediction. A prediction is a classification by IGTREE based on a local context of preceding words. In contrast with how IGTREE is used in the MBMT translation module, the word predictor classifier in WOPR produces class distributions (with more than one class if the classification occurs at a non-ending node).

Thus, WOPR usually returns more than one word for a given sequence, together with a probability based on frequency counts. This distribution of possible answers is used to calculate a perplexity value. There are three possibilities: (1) If the distribution returned by IGTREE contains the correct word, we take the probability of the word in the distribution; (2) If the distribution does not contain the correct word, we check if it is in the lexicon. If it is, the lexical probability is taken; (3) If it is not in the lexicon, a probability for unseen items is used that is estimated through Good-Turing smoothing. WOPR calculates the sum of $-p \log_2(p)$ of all the probabilities (one for each word in the sentence), and divides this by the number of words to obtain the average over the sentence. The perplexity value is two to the power of this sum.

Annotated Example

Besides its language modeling functionalities of predicting words and measuring perplexities, WOPR also provides the necessary tools to prepare the data, create datasets and train its prediction models. The following shows how a memory-based language model can be created starting from plain text data. Let us assume the file is called `corpus1.txt`. WOPR commands

generally have two parameters. The first one, `-r` tells WOPR which subroutine or tool to run. The second parameter, `-p` is a comma separated list of `keyword: value` pairs which specify the different parameters.

As a first step, WOPR is used to create a lexicon, which in this case is a list of words and their frequency in the corpus. It also generates a list with "counts of counts", which is used in Good-Turing smoothing of probabilities.

```
wopr -r lexicon -p filename: corpus1.txt
```

WOPR creates output file names based on the input file name and the command that is executed. In this case, it creates two files called `corpus1.txt.lex` and `corpus1.txt.cnt`. Next, we generate our windowed dataset. In this example we use a window size of three previous words. The resulting file is called `corpus1.txt.ws3`.

```
wopr -r window_s -p filename: corpus1.txt, ws: 3
```

We want to discard words with a frequency of five or less from our data set, and replace them with a special token <unk>. This is done with the following command:

```
wopr -r hapax -p filename: corpus1.txt.ws3, lexicon: corpus1.txt.lex,
hpx: 5
```

We then train our instance base. WOPR is used as a wrapper in this case, and most of the work is done by IGTREE. This could take some time, depending on the size of the data, but once the decision tree has been created and saved, it can easily be read in and used again.

```
wopr -r make_ibase -p corpus1.txt.ws3.hpx5, timbl: "-a1 +D"
```

Now we are ready to run our word predictor on a test file. The command to do this is as follows:

```
wopr -r pplxs -p filename: test1.txt.ws3.hpx5, ibasefile:
corpus1.txt.ws3.hpx5.ibase, timbl: "-a1 +D"
```

The test data is prepared in the same way as the training data. The following shows a line of output from WOPR. It shows an instance (`I would like`), the following word (`to`) and the –in this case correct– guess from the predictor, `to`.

```
I would like to to -0.351675 1.8461 1.27604 65 [ to 768 the 34
a 20 it 12 an 12 ]
```

This is followed by a number of statistics. The logprob of the prediction is -0.351675. The entropy of the distribution returned by IGTREE is 1.8461 ($-\sum p \log_2(p)$). The third number

shows the word level perplexity ($2^{-\mathrm{logprob}}$). The last number shows the number of elements in the distribution, in this case 65. This is followed by a top 5 of the distribution returned (with counts).

It is also possible to run WOPR in server mode, communicating over a socket connection with a client; in fact, this is how it is incorporated in the MBMT system. In server mode, WOPR will wait for a connection by another program and process the data it receives. The answer is sent back over the same connection.

5. Discussion

We have released MBMT, a straightforward translation model based on a fast approximation of memory-based classification. The approach fits into the EBMT framework; it models the mapping of sequences of word spans (here, word trigrams) in the source language to word trigrams in the output language. We showed that MBMT scales well to increased amounts of learning material. Within the current experiments we observed that training time and memory storage costs are approximately linear in the number of training examples. Translation speed on unseen data is very fast; our test set of 1,000 sentences was processed within seconds. Based on these results, we conclude for now that memory-based machine translation systems may be relevant in cases in which there is a need for fast and memory-lean training and/or classification. The low memory footprint may be additionally interesting for implementations of such systems in limited-capacity devices.

As a separate component of MBMT we have released the memory-based language model software package WOPR, which can also be used in isolation for general language modeling purposes. WOPR offers its functionality through command line options, but can also run in server mode; this is how MBMT uses WOPR.

Acknowledgments

This research is funded by NWO, the Netherlands Organisation for Scientific Research. We are grateful to Nicolas Stroppa, Andy Way, and Patrik Lambert for discussions and help with the comparison with Moses.

Bibliography

Bangalore, S., P. Haffner, and S. Kanthak. 2007. Statistical machine translation through global lexical selection and sentence reconstruction. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 152–159, Prague, Czech Republic. Association for Computational Linguistics.

Carl, M. and A. Way. 2003. *Recent Advances in Example-Based Machine Translation*, volume 21 of *Text, Speech and Language Technology*. Dordrecht, the Netherlands: Kluwer.

Carpuat, M. and D. Wu. 2007. Improving statistical machine translation using word sense disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 61–72.

Daelemans, W. and A. Van den Bosch. 2005. *Memory-based language processing*. Cambridge University Press, Cambridge, UK.

Daelemans, W., A. Van den Bosch, and A. Weijters. 1997. IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

Giménez, J. and L. Màrquez. 2007. Context-aware discriminative phrase selection for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 159–166, Prague, Czech Republic, June. Association for Computational Linguistics.

Och, F.-J. and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Stolcke, A. 2002. SRILM – An extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904, Denver, CO.

Stroppa, N., A. Van den Bosch, and A. Way. 2007. Exploiting source similarity for SMT using context-informed features. In A. Way and B. Gawronska, editors, *Proceedings of the 11th International Conference on Theoretical Issues in Machine Translation (TMI 2007)*, pages 231–240, Skövde, Sweden.

Van den Bosch, A. 2006. Scalable classification-based word prediction and confusible correction. *Traitement Automatique des Langues*, 46(2):39–63.

Van den Bosch, A., N. Stroppa, and A. Way. 2007. A memory-based classification approach to marker-based EBMT. In F. Van Eynde, V. Vandeghinste, and I. Schuurman, editors, *Proceedings of the METIS-II Workshop on New Approaches to Machine Translation*, pages 63–72, Leuven, Belgium.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 27–36

PostCAT – Posterior Constrained Alignment Toolkit

João Graça, Kuzman Ganchev, Ben Taskar

Abstract

In this paper we present a new open-source toolkit for statistical word alignments - Posterior Constrained Alignment Toolkit (PostCAT). The toolkit implements three well known word alignment algorithms (IBM M1, IBM M2, HMM) as well as six new models. In addition to the usual Viterbi decoding scheme, the toolkit provides posterior decoding with several flavors for tuning the threshold. The toolkit also provides an implementation of alignment symmetrization heuristics and a set of utilities for analyzing and pretty printing alignments. The new models have already been shown to improve intrinsic alignment metrics and also to lead to better translations when integrated into a state of the art machine translation system. The toolkit is developed in Java and available in source at its website [1]. We encourage other researchers to build on our work by modifying the toolkit and using it for their research.

1. Motivation

Word alignments are a valuable resource for several areas of natural language processing but especially for statistical machine translation (SMT) as they are a first step in most SMT pipelines. There has been a large quantity of research on improving word alignment models, the impact of different word alignments on SMT quality, and how to better use word alignments to extract the minimal units used in SMT systems such as phrases or rules. The vast majority of this work has unfortunately been done on in-house systems not released publicly with the result that researchers often have to re-implement previous work before they can improve upon it. The notable exception is the widely used GIZA++ toolkit (Och and Ney, 2003). Unfortunately there have been many improvements since the toolkit's publication, and the toolkit does not have many of the components for easy analysis of the alignment results.

We address this gap by introducing a new open-source toolkit - Posterior Constrained Alignment Toolkit (PostCAT) - that implements improved alignment models with results proven

[1] http://www.seas.upenn.edu/~strctlrn/CAT/CAT.html

Please cite this article as: João Graça, Kuzman Ganchev, Ben Taskar, PostCAT – Posterior Constrained Alignment Toolkit. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 27–36.

to boost SMT performance, as well as a set of utilities for the investigation of the effect of word alignment in overall SMT systems. The new models are trained with a new training procedure, called Constrained Expectation Maximization (Graça, Ganchev, and Taskar, 2008) that allows the user to specify prior information such as "alignments should be symmetric" or "each word should align to at most one word".

This toolkit has been used by us in three peer-reviewed publications. The procedure was introduced by Graça et al. (2008) and shown to improve intrinsic measures of alignment quality. Ganchev et al. (2008) show that the new word alignments and posterior decoding improve overall SMT quality on 6 different language pairs for different training sizes. The new models lead to an significant improvement (as measured by BLEU (Papineni et al., 2002)) in 16 out of 18 test cases. Finally the pretty printing and alignment statistics code was used in Graça et al. (2008) where golden sets of word alignments were produced for all combination of 4 different languages.

This paper is organized as follows. Section 2 gives a brief presentation of the models implemented in the framework. Section 3 describes the code organization, and some easy access points for improvement. Section 4 describes how to use the code to replicate the results in our previous papers. Section 5 describes related work and Section 6 concludes the paper.

2. Word Alignments Models

This section briefly describes the models implemented in the PostCAT. More detailed descriptions are available in (Brown et al., 1994), (Vogel, Ney, and Tillmann, 1996) and (Graça, Ganchev, and Taskar, 2008) along with derivations. Our goal here is to include enough details to make the next section easier to understand.

2.1. Baseline Models

The "baseline" models are the well know IBM Model 1, IBM Model 2 (Brown et al., 1994) and the HMM model proposed by (Vogel, Ney, and Tillmann, 1996). The three models can be expressed as:

$$p(\mathbf{t}, \mathbf{a} \mid \mathbf{s}) = \prod_j p_d(a_j|j, a_{j-1}) p_t(\mathbf{t}_j|\mathbf{s}_{a_j}), \tag{1}$$

with the three differing in their definition of the distortion probability $p_d(a_j|j, a_{j-1})$. Model 1 assumes that the positions of the words are not important and assigns uniform distortion probability. Model 2 allows a dependence on the positions $p_d(a_j|j, a_{j-1}) = p_d(a_j|j)$ and the HMM model assumes that the only the distance between the current and previous source word are important $p_d(a_j|j, a_{j-1}) = p_d(a_j|a_j - a_{j-1})$. All the models are augmented by adding a special "null" word to the source sentence. The likelihood of the corpus, marginalized over possible alignments is concave for Model 1, but not for the other models (Brown et al., 1994).

All these models we consider are normally trained using the Expectation Maximization (EM) algorithm (Dempster, Laird, and Rubin, 1977). The EM algorithm attempts to maximize the marginal likelihood of the observed data ($\mathbf{s}$, $\mathbf{t}$ pairs) by repeatedly finding a maximal lower

bound on the likelihood and finding the maximal point of the lower bound. The lower bound is constructed by using posterior probabilities of the hidden alignments ($\mathbf{a}$) and can be optimized in closed form from expected sufficient statistics computed from the posteriors. The posteriors are hardest to compute for the HMM alignment model but can be efficiently calculated by the forward-backward algorithm.

2.2. New Models

A well known problem when using the EM algorithm is the potential to be stuck in local maxima of the likelihood function. More importantly for word alignment, the models are very gross over-simplifications of the world and the optimal likelihood might not correspond to the optimal model parameters or optimal alignments. It has been shown that increases in likelihood can actually decrease alignment performance. The obvious solution to this problem is to bring the model closer to a faithful representation of the real world. This is the approach taken by IBM models 4+ (Brown et al., 1994) and by (Liang, Taskar, and Klein, 2006) who introduce an agreement component into the models with an intent similar to the one we address. Unfortunately, these changes can make the models intractable, requiring an approximation, often without any approximation guarantees. Graça et al. (2008) introduce an augmentation of the EM algorithm where the model remains unchanged, but the posteriors used during learning are constrained to be "meaningful." This has the advantage of allowing tractable inference while encoding prior knowledge that would be complicated to encode directly in the model.

The PostCAT provides training procedures that augment the baseline models by imposing constrains on the posteriors that roughly encode the following intuitions: "one word should not translate to many words" and "translation is approximately symmetric." We call the former "substochastic" and the latter "agreement." The class names in the PostCAT reflect this terminology. For example "SubstochasticM1" is IBM model one trained with the constraint that each source word should generate at most one target word in expectation. The original paper has a more detailed description of the constrained EM framework.

2.3. Decoding

For each sentence pair, the alignment models define a distribution over alignments. For use in an MT pipeline, we need to extract a single alignment from this distribution. The standard approach, called Viterbi decoding, is to choose the most probable alignment according to the model. Another possibility, that sometimes works better (Liang, Taskar, and Klein, 2006, Graça, Ganchev, and Taskar, 2008, Ganchev, Graça, and Taskar, 2008) is to include the alignment $i - j$ if the posterior probability that word i aligns to word j is above some threshold θ. This allows the accumulation of probability from several low-scoring alignments that agree on one point. This accumulated probability is easily extracted from the model posteriors. Note that this could potentially result in an alignment having zero probability under the model that generated it. PostCAT implements both decoding strategies with a variety of ways to tune θ, either by maximing/minimizing some intrinsic alignment metric such as alignment error rate

(AER (Och and Ney, 2003)) or (balanced) F-1 (Fraser and Marcu, 2007) with respect to a hand annotated corpus, or using a heuristic when aligned data is not available. One effective heuristic is to tune the threshold to have roughly the same number of points as Viterbi decoding. This trades less confident points for more confident ones.

2.4. Symmetrization

The word alignment models described above are asymmetric while most most applications including SMT require a single alignment for each sentence pair. Typically this gap is bridged by applying a symmetrization heuristic that takes as input two directional alignments and produces a single "consensus" alignment. The PostCAT implements the 4 most common alignment heuristics (Och and Ney, 2003): Intersection, Grow Diag, Grow Diag Final and Union. Should the user want another heuristic, implementing one in the PostCAT is very straightforward.

3. Code Organization

The code is organized around 4 main packages: The `corpus` that contains a representation of a sentence-aligned bilingual corpus, the `alignment` package containing a representation of word alignments, symmetrization heuristics, evaluation code as well as code to output alignments in machine-readable or human-readable form as well as collect statistics about alignments. The `models` package which contains the implementation the algorithms for training and extracting alignments from the models. Finally the `programs` package contains code to reproduce experiments from previous work, and example applications such as training and saving models, decoding with models and producing detailed reports of model performance. We describe each package in mode detail.

3.1. The `alignment` package

A word alignment, introduced by (Brown et al., 1994), consists of an object representing which words in a source language correspond to translations of other words in a target language, between two parallel sentences. Figure 1 shows an example of a word alignment represented as a matrix. An English sentence of length 8 (the rows of the matrix) is the translation of a Spanish sentence of length 9 (the columns of the matrix). Each entry of the matrix a_{ij} contains information about whether the i^{th} English word is the translation of the j^{th} Spanish word. The `Alignment` class stores this information and contains the identity of the respective sentences in the corpus and only contains the identities of the corresponding words. To save space, the `String` representations of the words are stored separately.

In addition to these identities, the `Alignment` class contains a matrix represent the word alignment. Each entry in the matrix takes one of several values indicating if the point from the gold standard, is obtained through decoding or has been added by a symmetrization heuristic. The `Alignment` class also stores posterior alignment probability of each word pair. This is

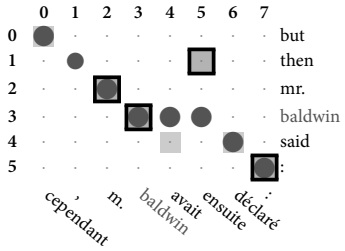

Figure 1. Example of a word alignment

useful for visualizations to understand how the alignment model works, as well as for more sophisticated phrase extraction techniques. In our pretty-printer posterior probabilities are represented by the size of the ball for each alignment entry. The class provides output methods for the most widely used word alignment formats (NACCL workshops, Giza++, Moses (Koehn et al., 2007)). Human readable output is implemented in `AlignerOutput` for plain text and `AlignerOutputLatex` for LaTeX. The latter was used to generate Figure 1.

The `AlignmentEvaluator` class implements the metrics for evaluating an alignment or group of alignments with respect to a gold standard. Currently implemented are precision, recall, AER , F-1, balanced F1 as well as methods to compute the number and type of unaligned points, useful in comparing different models. `AlignmentSymetrization` implements the symmetrization heuristics for word alignments: Intersection, Union, Grow Diagonal and Grow Diagonal Final.

3.2. The `corpus` package

The major class in the `corpus` package is `BilingualCorpus` which represents a parallel corpus, including any testing and development hand-aligned data. The relevant information about a corpus, such as file locations, existance of hand-aligned data and sentence length cutoffs is normally read from a plain-text configuration file. We found this to be much easier to use than passing this information piecemeal to our executables.

The class creates a dictionary for each language mapping string representations to integers to reduce the memory footprint of the program. It also collects statistics about individual words and creates a dictionary of word pairs that occur in the same sentence, also used for improved efficiency (both time and size) when training the models. There is also an option to load only part of the corpus for experiments with part of the training data.

3.3. The `model` package

The word alignment algorithms in the framework are implemented inside the `model` package. We implement IBM Model 1 and IBM Model 2 and the hidden Markov word alignment model, as well as constrained E-M version of these models. The constraints implemented

for the constrained E-M versions are "substochastic constraints" and "agreement constraints." Substochastic constraints capture the intuition that one word on one language should not align to many words in the other language (this mitigates the well known garbage collector effect (Brown et al., 1993)). Agreement constraints capture the intuition that alignment should be a roughly symmetric process so aligning with one language in source position should give the same results as aligning with that language in target position. Each of these 9 models is in a separate class. There are also some classes that efficiently represent model parameters that are shared between models. The models can be initialized with default parameters or with parameters from other models in order to boostrap complicated models from simpler ones. For example, it is standard practice to initialize the hidden Markov model with the translation table from IBM Model 1. The models can use smoothing add-n, specified at creation time.

3.4. The `programs` package

The `programs` package contains classes with `main` methods and can be used to reproduce the results reported in (Graça, Ganchev, and Taskar, 2008) and (Ganchev, Graça, and Taskar, 2008) without programming. Additionally there are illustrative examples of how to use the framework as a library. Some useful classes in the `programs` package:

- `SaveModels` - Trains a given model on a given corpus and save the trained model.
- `ComputeAlignmentError` - Loads a trained model and evaluates it against hand annotated data for a user-specified decoding scheme.
- `AlignmentsForMoses` - Loads a trained model and saves the alignments for different symmetrization heuristics in a format usable by the Moses script. To use Moses with our alignments, call the moses training script with `--firstStep 4`.
- `PrettyPrintAlignments` - Pretty prints the test set alignments for a given model and decoding scheme. We found this very useful for getting an understating of model behavior.

4. How to use PostCAT

This section describes how to use the PostCAT as a program to reproduce our results as well as how to extend it.

4.1. Getting and Installing the toolkit

The toolkit can be downloaded from its website as a gzipped tar file. It is implemented in Java and uses the GNU Trove library [2] which is distributed with the toolkit source code. The toolkit includes an Apache Ant[3] buildfile, so compiling it should just require running `ant`.

[2] http://trove4j.sourceforge.net/

[3] http://ant.apache.org/

4.2. Included Example

Included with the code is some sample data in the `small_data/` sub-directory. We have written some `bash` scripts as illustrations of how to use the code. To run these, you will need some auxiliary programs (e.g. `tee`, `find`, `tail`) that are standard on *NIX systems. If you don't have them, it should be easy to change the scripts or just copy the commandline. The small_data directory contains a corpus parameters file called `small_hansards.params`. In addition to the comments, the file has the following entries:

- Source and target language suffixes. For English, French we have 'en', 'fr'.
- A training, word-alignment development and test file bases. The files have '.en' appended for the English side and '.fr' for the French language.
- A name for the corpus. This is used when creating output directories.

We include scripts that we found useful when running our experiments. One script trains the hidden Markov alignment models (both baseline and agreement) and saves them. Another script computes alignment error metrics for each model. A third included script outputs alignments in a format accepted by the Moses SMT system. The package `README` file has more details on how to run the scripts and how to integrate the output with Moses.

To reproduce the results reported in (Graça, Ganchev, and Taskar, 2008) and (Ganchev, Graça, and Taskar, 2008) one needs to obtain the corpora separately.

4.3. Extending the toolkit

It is fairly straightforward to extend the toolkit with your own alignment model, or to produce statistics you might be interested in. In this subsection we will explain how to write the HMM with substochastic constraints, as an example for how to extend the toolkit with new models. We wrote a class `SubstochasticHMM extends HMM` to implement the model. In the interest of space we do not describe the implementation of the straightforward methods: the constructors and the methods to load and save the model, since their implementation is trivial. The main difference between the substochastic HMM and the regular HMM is that on the E-Step of the training method, we need to project the posteriors to a space where the sum of the posteriors for each source word is smaller than or equal to one. The mathematical derivation of the projection step is not central to the explanation of how to extend the toolkit so we include it as Appendix A. Suffice it to say that we can implement the projection step as a few iterations of gradient descent.

We created a copy of the `public EStepStats eStep()` method. After the computation of the posteriors (`makePosterior(forward, backward, likelihood)`) we project them onto the constraint set. The projection is implemented as the method

```
processPosteriors(posteriors, s, f, probCache)
```

where `s`, `f` and `probCache` are the source sentence, target sentence, and a cache of the translation probabilities respectively. The `MStep` method and inference methods are the same as in the original `HMM` so we do not need to implement them and we are done.

Suppose that having implemented our new model, we decide that we would like to per-

form a different kind of decoding. In particular, since we could perform the projection by computing some sentence-specific translation probabilities, we might want to try using these translation probabilities at decode time. If we want to do this, we would need to override the `posteriorDecodingAlignment` method of the `HMM` class. The new method will be almost identical to the old, with the exception that we would use the

```
processPosteriors(posteriors, s, f, probCache)
```

method that we implemented for `eStep()` in order to compute the projected posteriors right after the call to `makePosteriors`. To make a similar update to Viterbi decoding we need to override the `viterbiAlignment` method. At the start of the method, we would add code similar to the start of `posteriorDecodingAlignment` to compute posteriors and project them to the constraint set. We would then replace the call to `_tb.getProbability`, which currently uses the translation table to look up the translation probability and we would instead look up the translation probability in our `probCache` array, which has been updated by the call to `processPosteriors`.

5. Related Work

In a standard SMT pipeline PostCAT is a plug-in replacement for the GIZA++ toolkit and serves the same purpose. In fact if the goal is to produce word alignments using the baseline models we cannot recommend PostCAT over GIZA++ since the GIZA++ implementations are faster, and GIZA++ is integrated into most MT systems scripts (e.g. Moses [4], Syntax Augmented Machine Translation (SAMT) (Zollmann and Venugopal, 2006) [5]). Having said that, if the goal is to analyze the alignment results to understand how they impact translation performance, PostCAT provides useful visualization and analysis tools. Additionally, it provides a set of alignment algorithms, not implemented elsewhere and proven to work well for SMT. Furthermore, the code is easy to read and modify so if a researcher wants to extend the models, try different decoding or symmetrization schemes, or fine-tune some aspect of alignment this should be easy to do in PostCAT. For instance, two recent trends are the use of n-best alignments, and the use of alignment posterior probabilities when extracting phrases. Both of these are easily done in PostCAT since the posteriors are available within the `Alignment` object. Another open source word aligner, the Berkeley Aligner, available online[6] contains the implementation from (Liang, Taskar, and Klein, 2006). Their contribution is a model that has a similar intuition as our agreement constraints, but with very different realization. They define an intractable joint model and use an approximation to optimize model parameters.

[4] `http://www.statmt.org/moses/`

[5] `http://www.cs.cmu.edu/~zollmann/samt/readme.html`

[6] `http://www.cs.berkeley.edu/~pliang/software/`

6. Conclusions and Future Work

We have presented the Posterior Constrained Alignment Toolkit (PostCAT), an open-source toolkit for word alignment. Ongoing work on the toolkit is in three directions. Firstly the toolkit is being extended to work over the map reduce paradigm for parallelization. Secondly, new alignment models are being developed in the toolkit's framework and will be available in future versions. Thirdly, we are integrating new methods for phrase extraction, and rule extraction from a word aligned corpora, using the full information available in the word alignments.

Acknowledgments

J. V. Graça was supported by a fellowship from Fundação para a Ciência e Tecnologia (SFRH/ BD/ 27528/ 2006). K. Ganchev was supported by ARO MURI SUBTLE W911NF-07-1-0216.

Bibliography

Brown, P. F., S. A. Della Pietra, V. J. Della Pietra, M. J. Goldsmith, J. Hajic, R. L. Mercer, and S. Mohanty. 1993. But dictionaries are data too. In *Proc. HLT*.

Brown, Peter F., Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1994. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Royal Statistical Society, Ser. B*, 39(1):1–38.

Fraser, Alexander and Daniel Marcu. 2007. Measuring word alignment quality for statistical machine translation. *Comput. Linguist.*, 33(3):293–303.

Ganchev, Kuzman, João V. Graça, and Ben Taskar. 2008. Better alignments = better translations? In *Proc. of ACL-08: HLT*, Columbus, Ohio. Association for Computational Linguistics.

Graca, Joao, Joana Paulo Pardal, Luisa Coheur, and Diamantino Caseiro. 2008. Building a golden collection of parallel multi-language word alignment. In *LREC'08*.

Graça, Joao, Kuzman Ganchev, and Ben Taskar. 2008. Expectation maximization and posterior constraints. In *Proc. NIPS*.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL*. The Association for Computer Linguistics.

Liang, Percy, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *Proc. HLT-NAACL*.

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

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proc. ACL*.

Vogel, Stephan, Hermann Ney, and Christoph Tillmann. 1996. Hmm-based word alignment in statistical translation. In *Proc. COLING*.

Zollmann, Andreas and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *WSMT*. Association for Computational Linguistics, June.

A. Derivation of Substochastic Projection

For notational convenience we encode the desired constraints in terms of some feature functions $f_i(\mathbf{x}, \mathbf{a})$ for each source word i, where $\mathbf{x}$ are the two sentences and $\mathbf{a}$ is an alignment. Each feature is associated with a source word i. The feature is a function from an alignment $\mathbf{a}$ to a real number, counting how many foreign words are aligned to source word i in that particular alignment. We group all features f_i into a vector $\mathbf{f}$. Note that there are only linearly many such feature functions, even though we have exponentially many possible alignments $\mathbf{a}$. The projection step is the solution to the optimization problem:

$$\arg\min_{q} \mathrm{KL}(q(\mathbf{a}) \,||\, p_\theta(\mathbf{a}|\mathbf{x})) \;\; \text{s.t.} \;\; \mathbf{E}_q[\mathbf{f}(\mathbf{x}, \mathbf{a})] - 1 \leq 0. \tag{2}$$

where KL denotes Kullback-Leibler divergence, p_θ is the current model (with parameters θ) and $\mathbf{E}_q$ denotes expectation with respect to the probability distribution q. For more details of why we might want to do this we refer the reader to Graça et al. (2008). The optimization problem in Equation 2 can be efficiently solved in its dual formulation:

$$\arg\max_{\lambda \leq 0} \lambda^\top \mathbf{1} - \log \sum_{\mathbf{a}} p_\theta(\mathbf{a} \mid \mathbf{x}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\} \tag{3}$$

where we have solved for the primal variables q as:

$$q_\lambda(\mathbf{a}) = p_\theta(\mathbf{a} \mid \mathbf{x}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\}/Z, \tag{4}$$

with Z a normalization constant that ensures q sums to one. We have only one dual variable λ_i per constraint, and we optimize them by taking a few projected gradient steps. The partial derivative of the objective in Equation 3 with respect to parameter λ_i is simply $1 - \mathbf{E}_{q_\lambda}[\mathbf{f}_i(\mathbf{x}, \mathbf{a})]$. So we have reduced the problem to computing expectations of our features under the model q. For our particular features this reduces to computing expectations under the normal HMM model. To see this, we have by the definition of q_λ and p_θ,

$$\begin{aligned} q_\lambda(\mathbf{a}) = \;& \overrightarrow{p}(\mathbf{a} \mid \mathbf{x}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\}/Z \\ = \;& \prod_j \overrightarrow{p}_d(a_j | a_j - a_{j-1}) \overrightarrow{p}_t(\mathbf{t}_j | \mathbf{s}_{a_j}) \exp\{\lambda^\top \mathbf{f}(\mathbf{x}, \mathbf{a})\}/Z \\ = \;& \prod_{j, i = a_j} \overrightarrow{p}_d(i | i - a_{j-1}) \overrightarrow{p}'_t(\mathbf{t}_j | \mathbf{s}_i) \end{aligned}$$

Where we have let $\overrightarrow{p}'_t(\mathbf{t}_j|\mathbf{s}_i) = \overrightarrow{p}_t(\mathbf{t}_j|\mathbf{s}_i)e^{\lambda_i}$, and retained the same form for the model. So the projection step just creates some sentence-specific translation probabilites for each word pair. The inference procedures are other unchanged but use these updated translation probabilites. We enforce the constraint that $\lambda \leq 0$ by projecting to the negative orthant after each gradient step.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 37–46

An Open Source Rule Induction Tool for Transfer-Based SMT

Yvette Graham, Josef van Genabith

Abstract

In this paper we describe an open source tool for automatic induction of transfer rules. Transfer rule induction is carried out on pairs of dependency structures and their node alignment to produce all rules consistent with the node alignment. We describe an efficient algorithm for rule induction and give a detailed description of how to use the tool.

1. Introduction

Statistical Machine Translation (SMT) using deep linguistic representations for transfer is a relatively new and growing research area (Bojar and Hajič, 2008, Graham, 2008, Bojar et al., 2007a, Bojar and Čmejrek, 2007b, Graham et al., 2007, Ding and Palmer, 2006, Riezler and Maxwell, 2006, Ding and Palmer, 2005, Ding and Palmer, 2004a, Ding and Palmer, 2004b, Cmejrek et al., 2003, Eisner, 2003, Ding and Palmer, 2003, Hajič et al., 2002, Alshawi et. al., 2000a, Alshawi et. al., 2000b, Alshawi et. al., 1998). Training requires highly efficient algorithms; the training data is hierarchical dependency graphs as opposed to surface form stings and is therefore more complex than training data used for other methods of SMT (Brown et al., 1993); large numbers of rules (that contain a lot of morphological information) are needed to achieve high coverage of unseen data and rich statistical information. We provide a tool that uses an efficient algorithm for rule induction and outputs the rules in efficient O(n) size data structures (Graham, 2008). The rule induction tool, the parser/generator engine XLE [1] and transfer decoder [2] constitute a complete Transfer-Based SMT system.

[1] "XLE is available to a limited number of researchers ... For more information about obtaining a license, please contact thking at parc.com."

[2] The transfer decoder is currently in development and will be released as an open source tool in the near future.

Please cite this article as: Yvette Graham, Josef van Genabith, An Open Source Rule Induction Tool for Transfer-Based SMT. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 37–46.

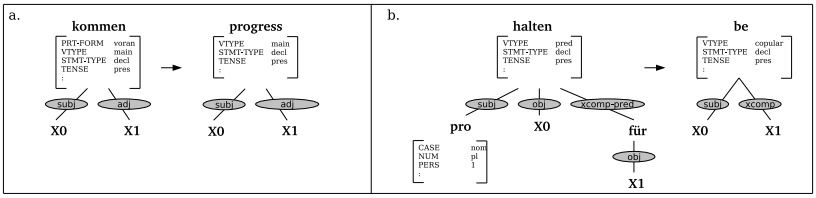

Figure 1. Example Transfer Rules

2. Transfer-Based SMT

Transfer-Based SMT is composed of three parts, i) parsing to linguistic structure, ii) transfer from SL linguistic structure to TL linguistic structure and iii) generation of TL sentence. Each step uses a statistical model to select the best output. For parsing, a disambiguation model is used to rank the parses and select the n-best output parses. Decoding (transfer) is then done on a parse structure (or n-best list of parse structures) via beam-search producing an n-best list of TL structures. For generation, the input is an n-best list of TL structures and all possible TL sentences are produced. The best TL sentence is then selected using an n-gram language model.

3. Training

The aim of rule induction is to acquire rules useful for transferring unseen SL structures by capturing correspondences between example training structure pairs of a parsed bilingual corpus. Figure 1(a) [3] shows a rule that captures an isomorphic German-English correspondence, while Figure 1(b) [4] captures a non-isomorphic correpsondence. Lexicalized nodes contain a vector of feature value pairs storing the morphological information belonging to that node.

3.1. Transfer Rule Induction

The transfer rule induction algorithm takes as input i) a dependency structure pair and ii) a one-to-one set of alignments between nodes of the dependency structure pair. Any alignment method can be used to align the nodes. For example, we currently use Giza++ (Och et al., 1999) for node alignment by constructing a bilingual training corpus from the predicate lemmas of the dependency structure nodes of the parsed bitext. Word alignment can then be run, in both

[3] The LHS contains a single lexicalized node with predicate value *(voran)kommen* with two arguments, a *subject* (X_0) and an *adjunct* (X_1) and the the RHS has the same structure but with *progress* as the root node.

[4] The LHS has the German predicate *halten* as its root node with three arguments, a *subject* (the lexicalized node with predicate *pro*), an *object* (X_0) and an *xcomplement-predicate* (the lexicalized node with predicate *für* and *object* X_1) and the RHS of the rule has *be* as its root node with two arguments, a *subject* (X_0) and an *xcomplement* (X_1).

language directions, as in Phrase-Based SMT (Koehn et al., 2003), followed by symmetrisation of the word alignments. We currently use the intersection of the word alignments as this gives a reliable set of one-to-one alignments.

3.1.1. Consistent Transfer Rules

As in Phrase-Based SMT, where a word alignment for each example sentence pair is first established before phrases consistent with that word alignment are extracted (Och et al., 1999, Koehn et al., 2003), we induce transfer rules that are consistent with the node alignment. We define a consistent transfer rule using a simplification of the actual training dependency structures and temporarily consider them as tree structures by ignoring edges that cause cycles in the graph or edges that share an end node with another edge. [5] Definition 1 applied to a (simplified) dependency structure pair yields a set of rules containing no variables by constraining rule induction using both the alignments between nodes and the position of the nodes within the two structures:

Definition 1.
Given a one-to-one set of alignments A between nodes in dependency pair (F, E), $(\overline{f}, \overline{e})$ is a rule consisting of nodes (N_f, N_e), rooted at (r_f, r_e), with descendents (D_f, D_e) of r_f and r_e in F and E respectively, if

$$
\begin{aligned}
& N_f = r_f \cup D_f \\
\bigwedge \quad & N_e = r_e \cup D_e \\
\bigwedge \quad & \forall f_i \in N_f : (f_i, e_j) \in A \rightarrow e_j \in N_e \\
\bigwedge \quad & \forall e_j \in N_e : (f_i, e_j) \in A \rightarrow f_i \in N_f \\
\bigwedge \quad & \exists e_j \in N_e : (r_f, e_j) \in A \\
\bigwedge \quad & \exists f_i \in N_f : (f_i, r_e) \in A
\end{aligned}
$$

Definition 2.
For any rule $(\overline{f}, \overline{e})$ in dependency pair (F, E) rooted at (r_f, r_e) consisting of nodes N_f and N_e, where $(\overline{s}, \overline{t})$ is also a rule in (F, E) rooted at (r_s, r_t) consisting of nodes N_s and N_t where $r_s \neq r_f$, $r_t \neq r_e$, if $r_s \in N_f$ and $r_t \in N_e$ then there is a rule $(\overline{a}, \overline{b})$ rooted at (r_f, r_e) with nodes r_s and r_t replaced by variable x_k, where k is an index unique to the transfer rule, consisting of nodes:

$$N_a : N_f \backslash N_s \cup x_k$$
$$N_b : N_e \backslash N_t \cup x_k$$

To help visualize what is considered a consistent transfer rule, Figure 2(b) shows the example dependency structure of Figure 2(a) divided into parts by a number of boxes with corresponding parts of the dependency structure pair labelled with the numbers 1-6. Each consistent transfer rule can be realized by assigning a true or false value to each pair of boxes, so that

[5] For example, if the subject of node A is also the subject of node B, one of these these edges is ignored temporarily. This is done by labelling the nodes using an increasing index in depth first order (only labelling each node once). Edges with an end node label less than their start node are ignored.

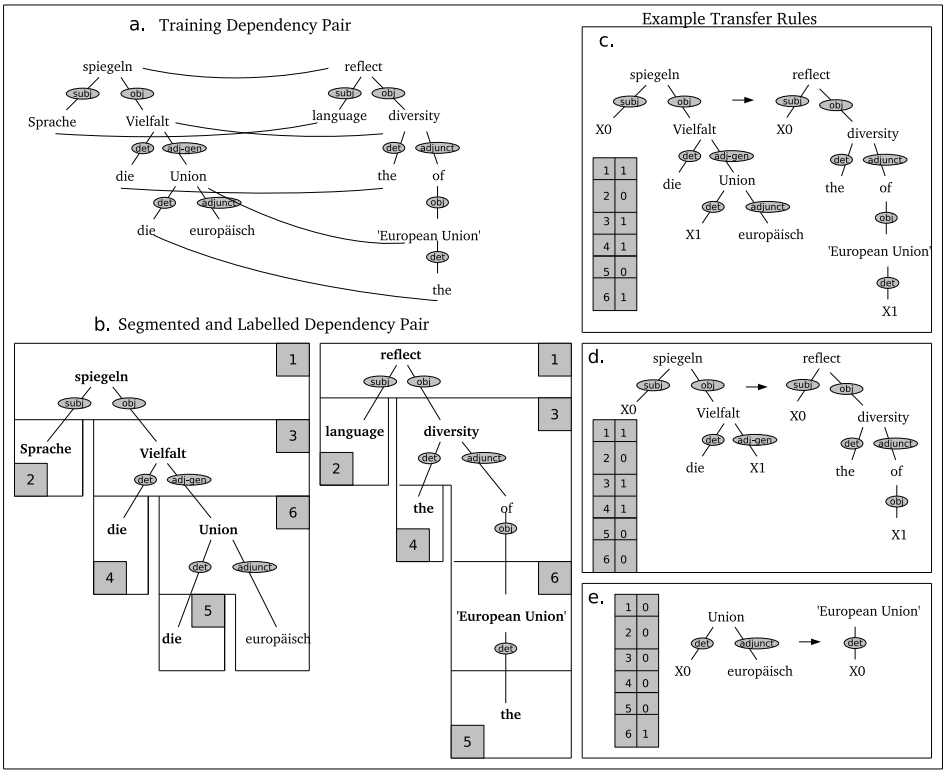

Figure 2. Consistent Transfer Rules

boxes assigned true are included in the rule and boxes assigned false are left out. Combinations of true and false values are constrained and this can be visualized by only allowing adjoining boxes in Figure 2(b) to be labelled true for any one rule. Figures 2(c), 2(d) and 2(e) show consistent rules with truth value combinations. According to Definition 1, two nodes may form the root of a transfer rule if they both have the same non-empty set of aligned descendents (rule root nodes are in bold in Figure 2(b)). This ensures that the entire subtree rooted at the SL root of a transfer rule corresponds to the entire subtree rooted at the TL root. In addition, the root nodes of a rule must each be an aligned node, i.e. each must be aligned to some node in the other side of the rule. This ensures that unaligned nodes do not form the root of a transfer rule and are, therefore, only included in rules that also contain their head node, giving them a meaningful context. For example, the rule in Figure 2(e) is allowed by Definition 1, but a rule with this LHS and a RHS rooted at *of* is disallowed. The node *of* does not have a lexical correspondent in the German structure, but instead its head *diversity*, the dependency label *adjunct*

and node *of* together correspond to *Vielfalt* and the dependency label *adjunct-gen* (genative adjunct). Definition 2 describes how rules with variables are produced by replacing both sides of a nested rule, i.e. a rule nested inside a larger rule, with a variable X_n. For example, in Figure 2, rule (d) was produced from rule (c) by replacing the nested rule (e) with variable X_1. Introducing variables to transfer rules in this way potentially produces a very large number of rules. In the worst case, when we have isomorphic structures with all nodes aligned, the number of rules is exponential. However, in reality, dependency structures of real world parallel corpora are very rarely entirely isomorphic with a very high number of aligned nodes. If the number of rules induced does indeed become unmanageable, however, rule induction can be constrained further by putting a limit on the size of rules. [6]

3.2. Transfer Rule Induction Algorithm

The transfer rule induction algorithm works by encoding all consistent rules of the SL and TL dependency structure pair in a single structure. The most complex part of the algorithm decides which node pairs within the dependency structure pair form *rule roots*. Once the *rule roots* of the dependency structure pair have been decided, the entire set of SL and TL dependency triples are then simply recorded with each dependency triple slightly modified by labelling it with a *context variable* (A_i) (Maxwell and Kaplan, 1991) and by replacing the original node labels with variables (X_i). Labelling the dependency triples with *context variables* allows us to encode all rules consistent with the node alignment in a single structure. This method of encoding allows $O(2^n)$ rules to be encoded in an O(n) size structure and is described in detail in Graham and van Genabith (2008). The algorithm for choosing the *rule root* nodes of the dependency structure pair is given in Figure 3. [7] The complexity of the algorithm is $O(a^2 \log a)$ in the worst case, where a is the number of aligned node pairs.

4. Using the Rule Induction Tool

The rule induction tool requires a Prolog engine to run. The system has been developed and tested with SICStus Prolog. Included in the download package are 4000 German-English sample dependency structure pairs from a portion of the Europarl Corpus (Koehn et al., 2005). [8] The sample dependency structures are divided into sets, each containing 1000 sentences.

[6] We currently do not limit the size of rules as we train on a restricted sentence length of 5 to 15 words. This results in an average number of rules induced per sentence pair of 32.47, with average sentence length 9.89 for German and 10.48 for English.

[7] The program itself is written in Prolog, and uses some of Prolog's built-in features that are not available in other programming languages. To keep the pseudocode as implementation independent as possible, when we use a Prolog specific function or control structure we describe it in pseudocode using an equivalent function or control structure available in most programming languages. For example, Prolog has a built in indexing of terms, that uses the first argument of the term as a key to achieve an O(log n) return time when searching for that term in memory. We use this in our Prolog implementation but where we do so we described it in pseudocode as using a hash table.

[8] The sample sentences were parsed with XLE parse engine (Kaplan et al., 2002) to Lexical Functional Grammar (Kaplan and Bresnan, 1982, Bresnan, 2001, Dalrymple, 2001) f-structures using German and English grammars (Rie-

```
Algorithm RuleRoots( List sl_nodes, List tl_nodes,
                     HashTable <sl_node_id,alignment_id> sl_alignments,
                     HashTable <tl_node_id,alignment_id> tl_alignments):

  # For each aligned SL node create a list
  # containing the alignment ids of its aligned descendents
  # Put lists in a Hash Table
  sl_aligned_descs = new HashTable<list_of_aligned_descs,sl_node_id>
  foreach  s ∈ S
    if exists sl_alignments.get(s.node_id) then
      list = new empty list
      foreach d ∈ descendents(s)
        if exists sl_alignments.get( d.node_id) then
          a_id = sl_alignment.get( d.node_id)
          list.add( a_id)
      sl_aligned_descs.put( list, s)

  # Likewise for TL nodes
  tl_aligned_descs = new HashTable<list_of_aligned_descs,tl_node_id>
  foreach  t ∈ T
    if exists tl_alignments.get(t.node_id) then
      list = new empty list
      foreach d ∈ descendents(t)
        if exists tl_alignments.get( d.node_id) then
          a_id = tl_alignment.get( d.node_id)
          list.add( a_id)
      tl_aligned_descs.put( list, t)

  # Find node pairs with matching sets of aligned descendents
  roots = new empty List
  foreach key in keys( sl_aligned_desc)
    if exists tl_aligned_descs.get( key)
      # A pair has been found
      i = sl_aligned_descs.get( key)
      j = tl_aligned_descs.get( key)
      roots.add( i, j)
  return roots
```

Figure 3. Algorithm to choose the rule roots *in the SL and TL dependency structures*

1. Download the package from
   http://www.computing.dcu.ie/~ygraham/software.html
2. Unzip ria.tar.gz and extract the files. This should produce a folder called *ria*.
3. Create an environment variable called RIA and set it to the location of the tool, for example: RIA=/home/jsmith/ria; export RIA;
4. Add the location of the bin directory to your PATH environment variable, for example: PATH=$PATH:$RIA/bin; export PATH;
5. Test the tool by typing the following command to induce all rules from set 0 of the sample training sentences: ria 0

A file containing the rules of set 0 should now be written in the following directory: $RIA/output/rules.

4.1. Output Format

The rules induced from each training dependency pair are stored in a file containing the rules (in a single compact size structure), and a file with some additional information about the rules. For example, the following two files store rules for sentence 123 of set 0:

- $RIA/output/rules/sents_0000/R123.pl
- $RIA/output/rules/sents_0000/I123.pl

To retrieve the rules, both of these files should be loaded by SICStus before calling the following predicate:

- transfer_rule(+-S, +-T, +-Fs_id, +-Root_id, -LHS, -RHS, +-Options).

We include an example program [9] that retrieves all rules for a specified sentence pair and records them:

- write_rules 123

The enumerated rules for sentence id 123 should now be written in:

- $RIA/output/example/R123

4.2. Running the Tool on New Training Data

Start by converting the dependency structures into the same Prolog format of the sample structures.[10] The training structures should be divided into sets of 1000 and put in directories in the following locations:

- $RIA/data/sl_train
- $RIA/data/tl_train
- $RIA/data/alignments

---

zler et al., 2002, Butt et al., 2002). For node alignment, Giza++ (Och et al., 1999) was run in both language directions and the intersection was gotten using moses (Koehn et al., 2007).

[9] The rules should be retrieved by loading the rule and information files and then calling findall on transfer_rules/7, as is done in the example program, as it is not necessary to enumerate all rules on disk to use them.

[10] The sample structures are in the output format of the XLE parse engine (for further details see http://www2.parc.com/isl/groups/nltt/xle/doc/xle.html#Prolog_Output).

For example, SL and TL dependency structures for sentence id 134067 should be put in two separate files;

- $RIA/data/sl_train/sents_0134/S067.pl and
- $RIA/data/tl_train/sents_0134/S067.pl,

and node alignments in a file called

- $RIA/data/alignments/sents_0134/A067.pl.

These files can then be archived and zipped. Name them as follows:

- $RIA/data/sl_train/sents_0134.tar.gz
- $RIA/data/tl_train/sents_0134.tar.gz
- $RIA/data/alignments/sents_0134.tar.gz

Rule induction can then be run for this set of structures using the command:

- ria 134

5. Conclusion

We presented an open-source tool for automatic transfer rule induction from parsed bilingual corpora. We described an efficient algorithm that induces transfer rules from dependency structure pairs encoding rules in an efficient O(n) size data structure (Graham, 2008). We hope that this tool is used to produce interesting research.

Acknowledgments

Many thanks to John Maxwell, Joachim Wagner, Marcus Furlong, Mary Hearne, Philipp Koehn and our reviewers.

Bibliography

Hiyan Alshawi, Srinivas Bangalore and Shona Douglas. 1998. Automatic Acquisition of Hierarchical Transduction Models of Machine Translation. In *Proceedings of COLING-ACL 1998*.

Hiyan Alshawi, Srinivas Bangalore and Shona Douglas. 2000. Learing Dependency Translation Models as Collections of Finite State Head Transducers. In *Computational Linguistics*, 26(1):45-60.

Hiyan Alshawi and Shona Douglas. 2000. Learning Dependency Transduction Models from Unannotated Examples. In *Philisophical Transactions A*, The Royal Society 358:1357-1372.

Ondřej Bojar, Silvie Cinkova and Jan Ptaček. Towards English-to-Czech MT via Tectogrammatical Layer. In *Proceedings of the Sixth International Workshop on Treebanks and Linguistic Theories (TLT 2007)*, Bergen, Norway, December. 2007

Ondřej Bojar and Martin Čmejrek. 2007. Mathematical Model of Tree Transformations. *Project Euromatrix Deliverable 3.2*, Ufal, Charles University, Prague.

Ondřej Bojar and Jan Hajič. 2008. 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.

Joan Bresnan. 2001. *Lexical-Functional Syntax.*, Blackweelm Oxford, 2001.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2): 263-311.

Miriam Butt, Helge Dyvik, Tracy H. King, Hiroshi Masuichi and Christian Rohrer. 2002. The Parallel Grammar Project. (grammar version 2005) In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02), Workshop on Grammar Engineering and Evaluation*, pages 1-7. Tapei, ROC.

Martin Cmejrek, Jan Curín and Jirí Havelka 2003 Czech-English Dependency Tree-based Machine Translation In *Proceedings of EACL 2003*, pages 83-90.

Mary Dalrymple. *Lexical Functional Grammar*, Academic Press, San Diego, CA; London. 2001.

Yuan Ding, Daniel Gildea and Martha Palmer. 2003. An Algorithm for Word-Level Alignment of Parallel Dependency Trees. In *Proceedings of the Machine Translation Summit 2003*.

Yuan Ding and Martha Palmer. 2004. Automatic Learning of Parallel Dependency Treelet Pairs. In *Proceedings of the 1st International Joint Conference on Natural Language Processing*.

Yuan Ding and Martha Palmer. 2004. Synchronous Dependency Insertion Grammars A Grammar Formalism for Syntax-Based Statistical MT. In *Proceedings of COLING 2004*.

Yuan Ding and Martha Palmer. 2005. Machine Translation Using Probabilistic Synchronous Dependency Insertion Grammars. In *Proceedings of the 43rd Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 541-548, Ann Arbor, June 2005.

Yuan Ding and Martha Palmer. 2006. Better Learning and Decoding for Syntax Based SMt Using PSDIG. In *Proceedings of AMTA 2006*

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 205-208, Sappora, July 2003.

Yvette Graham, Deirdre Hogan and Josef van Genabith. 2007. Automatic Evaluation of Generation and Parsing for Machine Translation with Automatically Acquired Transfer Rules. In *Proceedings of the 2007 Workshop on Using Corpora for NLG: Language Generation and Machine Translation*, at MT Summit XI, Copenhagen, September 2007.

Yvette Graham and Josef van Genabith. 2008. Packed Rules for Automatic Transfer Rule Induction. In *Proceedings of the European Association of Machine Translation Conference 2008*, Hamburg, Germany.

Jan Hajič, Martin Čmejrek, Bonnie Dorr, Yuan Ding, Jason Eisner, Daniel Gildea, Terry Koo, Kristin Parton, Gerald Penn, Dragomir Radev and Owen Rambow. 2002. Natural Language Generation in the Context of Machine Translation. Technical Report. *NLP WS'02*, final report.

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

Ronald Kaplan and Joan Bresnan. 1982. Lexical Functional Grammar, a Formal System for Grammatical Represenation. In Bresnan, J. editor, *The Mental Representation of Grammatical Relations*, pages 173-281, MIT Press, Cambridge, MA.

Philipp Koehn, Franz Josef Och and Daniel Marcu. 2003. Statistical Phrase-based Translation. In *Proceedings of the HLT-NAACL 2003*, pages 48-54, Edmonton, May/June 2003.

Philipp Koehn. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of MT Summit 2005*

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison Burch, Richard Zens, Alexandra Constantin, Marcello Federico, Nicola Bertoldi, Chris Dyer, Evan Herbst, Brooke Cowen, Wade Shen, Christine Moran and Ondřej Bojar. 2007. Moses: Open Source Toolkit for Statistical Machine Translation In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*

John T. Maxwell III and Ronald M. Kaplan. 1991. A Method for Disjunctive Constraint Satisfaction. In *Current Issues in Parsing Technology*, Masaru Tomita editor, pages 173-190, Kluwer Aca demic Publishers.

Franz Josef Och, Christoph Tillmann Hermann and Ney. 2000. Improved Alignment Models for Statistical Machine Translation. In *Proceedings of the 1999 Conference on Empirical Methods in Natural Language Processing* (EMNLP'99). College Park, MD, pages 20-28.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using Lexical Functional Grammar and discriminitive estimation techniques . (grammar version 2005) In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL)*, Philadelphia, July 2002.

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

PBML

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

Decoding in Joshua

Open Source, Parsing-Based Machine Translation

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

Abstract

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

1. Motivation

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

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

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

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

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

2. Parsing-based MT Decoder

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

2.1. Grammar Formalism

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

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

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

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

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

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

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

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

2.2. MT Decoding as Chart Parsing

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

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

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

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

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

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

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

2.3. Pruning in a Decoder

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

2.4. Hyper-graphs and k-best Extraction

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

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

3. Underlying Methodologies

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

3.1. Extendibility

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

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

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

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

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

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

3.2. End-To-End Cohesion

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

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

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

3.3. Scalability

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

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

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

4. Using the Decoder

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

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

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

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

5. Experimental Results

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

5.1. System Training

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

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

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

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

# lm model weight
lm 1.000000

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

# wordpenalty weight
wordpenalty -2.844814
```

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

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

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

5.2. Improvements in Decoding Speed

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

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

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

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

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

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

decoder is stable and scalable.

5.3. Impact of a Distributed Language Model

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

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

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

6. Conclusions

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

Acknowledgments

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

Bibliography

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

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

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

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

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

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

Galley, Michel, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL-CoLing-2006)*, Sydney, Australia.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 57–66

apertium-cy – a collaboratively-developed free RBMT system for Welsh to English

Francis Tyers, Kevin Donnelly

Abstract

apertium-cy (http://www.cymraeg.org.uk) is a rule-based "gisting" machine translation system for Welsh to English, with both engine and data released under the GPL. We summarise the development of *apertium-cy*, evaluate its output, and discuss the advantages of a collaborative development model combined with rule-based MT for marginalised languages.

1. The Apertium platform

apertium-cy is a "gisting" machine translation system for Welsh to English, based on the Apertium machine translation platform.[1] The platform was originally aimed at the Romance languages of the Iberian peninsula, but is now being adapted for other languages (such as Basque, and languages from the Celtic group – Welsh, Irish, Breton), with much of the work on new languages being pursued by volunteers, following the increasingly common collaborative development model used for free[2] and open-source software. Apertium is licensed under the Free Software Foundation's GNU General Public License,[3] and all the software and data for the 17 supported language pairs (and the other pairs in development) is available for download from the project website.

Apertium follows a shallow-transfer approach, and is very fast. Finite-state transducers (Garrido-Alenda and Forcada, 2002, Roche and Schabes, 1997) processing up to 40,000 words per second are used for lexical processing, first-order hidden Markov models (HMM) are used for part-of-speech tagging, and multi-stage finite-state based chunking for structural transfer.

[1] http://www.apertium.org

[2] We follow the definition of "free" used by the Free Software Foundation - http://www.fsf.org.

[3] http://www.fsf.org/licensing/licenses/gpl.html, accessed 12/12/2008.

Please cite this article as: Francis Tyers, Kevin Donnelly, *apertium-cy* – a collaboratively-developed free RBMT system for Welsh to English. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 57–66.

The software behind the platform is implemented as a standard UNIX pipeline, with each stage in the translation having a separate C++ program. Communication between each stage uses piped text streams. XML-based formats are used to encode the linguistic data, which are then compiled into the high-speed formats used by the engine. Further details are given in (Armentano-Oller et al., 2006), and on the project website.

2. Background

The increasing economic and cultural importance of information technology poses a threat to marginalised languages. Unless they have an ICT presence, along with a reasonable range of the tools that computer-users take for granted (e.g. spelling and grammar-checkers, dictionaries and thesauruses, etc.), they run the risk of being shut out of this whole sector of modern life. The result will be a further decline in their status and usage (Crystal, 2000).

In the case of Welsh, which could be said to be the healthiest of the Celtic languages, with nearly 600,000 speakers (almost 21% of the population),[4] requests had been made over a period of years for leading software suppliers to produce Welsh versions of their software, but little progress had been made. Free software offered a way around this problem.

In 2003, the second author launched a voluntary initiative to translate software used on the GNU/Linux operating system into Welsh. Language tools that majority languages take for granted would have been useful to correct mistakes and maintain quality, but unfortunately, where such tools existed, they were not available under a licence which would enable them to be distributed with the software being translated (the exception being a Welsh spellchecker which was released some time later for the Firefox browser).

The only alternative was to create these tools from scratch. This involved delaying the translation project, and working in turn on a dictionary,[5] verb inflector,[6] and a grammar-checker based on *An Gramadóir* (Scannell, 2008).

3. Development of *apertium-cy*

When we started extending Apertium to create a gisting system for Welsh by using this free data, a major issue to be tackled was revising its dictionary format to deal with perhaps the defining feature of the Celtic languages – mutation (alteration of initial phonemes, usually having a morphological significance). An example would be the lemma *tad* (father), which can appear as *dad* (*ei dad* – his father), *nhad* (*fy nhad* – my father) and *thad* (*ei thad* – her father). For further details, see (Ball and Müller, 1992) and (Stewart, 2004).

It also became clear that there were two main issues with the disambiguation stage in the Apertium platform with respect to Welsh. The first was that the HMM-based POS tagger was

[4] Table WLP01 in: `http://www.statistics.gov.uk/downloads/census2001/Report_on_the_Welsh_language.pdf`, accessed 12/12/2008.

[5] `http://www.eurfa.org.uk`

[6] `http://www.konjugator.org.uk`

not able to take advantage of the disambiguation properties of the mutations, and the second was that the accuracy of a tagger trained in an unsupervised manner (without a tagged corpus) was below what was expected.

The following phrase gives an example of mutation having an effect on disambiguation: *Mae'r modd y mae gweithwyr mudol...* (*The means by which migrant workers...are...*). Here, *modd* could be interpreted as either an unmutated form of the noun *modd* (means) or as a nasally-mutated form of the noun *bodd* (pleasure). Linguistically there is no ambiguity – a nasal mutation would never follow the definite article, *[y]r*.

Since there was no tagged corpus of Welsh available under a free licence with which to train the tagger in a supervised manner, we examined other options for improving disambiguation performance. Rather than add this functionality directly to the Apertium tagger, the first author undertook a review of free software which might meet our needs here, and settled on Constraint Grammar[7], being developed by researchers at the University of Southern Denmark – further details are available in (Karlsson et al., 1995), and on the webpage referenced. It proved possible to integrate this software relatively easily, and a number of CG rules were written to disambiguate the Welsh text before it was fed into the statistical tagger.

Version 0.1 of *apertium-cy* contains approximately 10,000 lemmas and 150 grammatical rules, and has up to 94% coverage on large Welsh corpora.[8] Work continues on 0.2, with the key target being an initial version of an English to Welsh translator.

4. Evaluation

Previous work reported for Welsh machine translation includes (Phillips, 2001) and (Jones and Eisele, 2006). (Somers, 2004), in a report commissioned by the Welsh Language Board on a possible strategy for MT in Welsh, suggests (section 4.1) that one of the initial steps should be "a small Welsh-English...system for gist translation of Welsh documents". We would argue that *apertium-cy* constitutes such a system.

4.1. Quantitative

In order to provide a comparison with the other published paper on the subject, (Jones and Eisele, 2006), we evaluated the quality of the translations using three common metrics: word error rate (WER), position-independent word error rate (PER), and the Bilingual Evaluation Understudy (BLEU) (Papineni et al., 2002).

The first two metrics (WER and PER) provide a measure of post-edition effort – that is, they measure how much of the text needs to be changed to achieve a translation of publishable quality.

The third (BLEU) is a popular evaluation metric within the machine translation community, and we include it to give results comparable with the system described in (Jones and Eisele,

[7] `http://beta.visl.sdu.dk/constraint_grammar.html`, accessed 12/12/2008.

[8] Coverage for the PNAW corpus – see section 4.1 below – is 94%, and for the Welsh Wikipedia, 86%.

2006). It should be noted that BLEU scores are not necessarily appropriate for comparing systems of different types, and have a tendency to penalise rule-based systems (Callison-Burch, Osborne, and Koehn, 2006, Labaka et al., 2007).

Two corpora were used for evaluation. The first corpus consisting of 318 sentences (5,492 words) was selected at random from the Welsh Wikipedia - we have made this corpus available for download.[9] These sentences were translated and then post-edited by a speaker of both languages. We then calculated the WER, PER and BLEU scores of the system's translations against the post-edited references. It is worth noting that (a) sentences were filtered such that each one is covered fully by the dictionaries, with the result that this test does not give an indication of system vocabulary coverage, and (b) using BLEU against a post-edited reference is a novel methodology and may lead to higher scores.

The second corpus consisted of a 10% sample (50,000 sentences) of the Welsh–English Proceedings of the National Assembly for Wales (PNAW) corpus.[10] For this corpus, the sentence sample was translated, and then the scores were calculated between the translated sentences and the reference translations.[11] As with (Jones and Eisele, 2006), unknown words were left in the translations.

The results of these evaluations are given in Table 1 – bold face represents the highest score.

	WER	**PER**	**BLEU**
Wikipedia true-case	55.78	30.59	30.70
Wikipedia lowercased	**53.40**	**27.22**	**32.21**
PNAW true-case	65.99	35.44	15.12
PNAW lowercased	**64.94**	**34.35**	**15.68**

Table 1. WER, PER and BLEU metrics for the two corpora

As expected, considering the system is intended for "gisting translation", the scores for the post-edition task indicate that the system currently produces sentences with too high an error rate to be usable for professional post-editing.

There is less difference between the PER scores for each corpus than there is for the WER and BLEU scores. This probably indicates (unsurprisingly given the differences in sentence structure between Welsh and English) that more needs to be done on word-reordering, which this particular metric does not take into account.

It is also surprising that the BLEU score for the PNAW corpus is substantially lower than for the Wikipedia one, while the scores for WER and PER are less divergent. All of the BLEU scores are lower than the 40.22 for Welsh to English reported in (Jones and Eisele, 2006), but

[9] `http://xixona.dlsi.ua.es/~fran/welsh/cy-test-corpus.tar.gz`

[10] See (Jones and Eisele, 2006), available from `http://xixona.dlsi.ua.es/corpora/`.

[11] It is generally recommended that three references are used as a minimum for calculating BLEU scores. Unfortunately three references were not available, and (Jones and Eisele, 2006) reports using a single reference translation.

given the variation in the scores we suspect, along with the previously mentioned authors, that BLEU is not a particularly good metric for comparing unrelated MT systems.

4.2. Qualitative

The targets for *apertium-cy* 0.1 included the aim that "sentences of up to 5 words should be translated reasonably well from Welsh to English",[12] since this will allow a large number of short, conceptually-simple sentences to be translated. This has been met quite comfortably, as the following examples (with *apertium-cy* output in line b) show:

(1) a. *Mae'r gath yn yr ardd, ond mae'r ci yn y cae.*
b. The cat is in the garden, but the dog is in the field.

(2) a. *Mi welodd y dyn y bachgen yn dod allan o'r siop.*
b. The man saw the boy coming out of the shop.

(3) a. *Mi fydd y trên yn hwyr yfory, oherwydd bydd y cwmni yn gweithio ar y lein.*
b. The train will be late tomorrow, because the company will be working on the line.

Other sentences do not come across as well-formed English at present, and are therefore not suitable for dissemination, but the meaning is perfectly clear:

(4) a. *Bydd rhaid i ti frwsio dy ddannedd cyn mynd i'r ysgol.*
b. *Necessity will be to you brush your teeth before go to the school.
c. You'll have to brush your teeth before going to school.

(5) a. *Mi gafodd yr adroddiad ei olygu gan y Pwyllgor Seneddol.*
b. *The report got edit with the Parliamentary Committee.
c. The report was edited by the Parliamentary Committee.

apertium-cy 0.1 has been tested on official statements, novels, newspaper articles and non-fiction, and also seems to work on older texts provided the spelling is modernised, as in this example from 1865:

(6) a. *Ond oherwydd na ellid disgwyl ond cylchrediad lleol i lyfr o'r fath, ac oherwydd nad oedd gennym ninnau arian i'w gwario mewn ymgymeriad felly, ofnwyd yr anturiaeth.*
b. *But because nor could expect but local circulation to book of the type, and because was not with us us a money to spend it in undertaking so, the adventure were feared.
c. But since only a local circulation for a book of this kind could be expected, and because we ourselves did not have the money to spend on such an undertaking, the venture raised fears.

The web interface at `http://www.cymraeg.org.uk` gives other examples, and allows users to enter their own text in order to come to their own conclusions about translation quality.

[12] `http://wiki.apertium.org/wiki/Welsh_to_English`

5. Shortcomings

The main shortcomings of *apertium-cy* 0.1 fall into three main areas: phrase delimitation, treatment of subordinate clauses, and lexical selection.

5.1. Phrase delimitation

Because Apertium follows a shallow transfer approach, and does not include a full parser, it can be difficult to delimit phrases in such a way that they can be handled as a block when the target language requires this. Consider the following series of examples, based on the standard Welsh genitival construction, where the sequence *noun1 + def.art + noun2* is equivalent to the English *the noun1 of the noun2* or *the noun2's noun1*:

(7) a. *cath* *ddu*
[$_{NP}$ noun + qual]
b. black cat

(8) a. *cath* *y* *meddyg*
[$_{NP}$ noun + def.art + noun]
b. the cat of the doctor
c. the doctor's cat

(9) a. *cath* *ddu* *y* *meddyg*
[$_{NP}$ noun + qual + def.art + noun]
b. *black cat the doctor
c. the doctor's black cat

(10) a. *cath* *ddu* *fawr* *y* *meddyg*
[$_{NP}$ noun + qual + qual + def.art + noun]
b. *big black cat the doctor
c. the doctor's big black cat

(11) a. *cath* *merch* *y* *meddyg*
[$_{NP}$ noun + [$_{NP}$ noun + def.art + noun]]
b. *daughter cat the doctor
c. the doctor's daughter's cat

(12) a. *cath* *ddu* *merch* *y* *meddyg*
[$_{NP}$ noun + qual + [$_{NP}$ noun + def.art + noun]]
b. *daughter black cat the doctor
c. the doctor's daughter's black cat

(7) and (8) are well-formed English, and (9) and (10) are only missing a definite article and the genitival *of*. However, in (11), the sub-phrase *merch y meddyg* should be translated

along the lines of (8) as *the daughter of the doctor*, and left in position. Instead, *merch* is treated as a qualifier like *ddu*, and shifted, as can be seen when comparing (12) and (10). Most instances of problematic phrase delimitation are more complex than this, but we are confident that adding more rules and refining existing ones, along with the planned additon of a basic parser to Apertium some time in 2009, will resolve many of the issues.

5.2. Treatment of subordinate clauses

Marked formations and subordinate clauses, particularly relative clauses, also need more work:

(13) a. *Mi welodd y dyn y bachgen sy'n gweithio yn y siop.*
b. The man saw the boy that works in the shop.

(14) a. *Mi welodd y dyn y bachgen a giciodd y ci.*
b. *The man saw the boy and the dog kicked.
c. The man saw the boy who kicked the dog.

(15) a. *Hi yw'r ferch a welais ddoe.*
b. *She the daughter is and saw yesterday.
c. She is the girl I saw yesterday.

(16) a. *Ef yw'r dyn y lladdwyd ei fab.*
b. *He the man is were killed its son.
c. He is the man whose son was killed.

Part of this, as in (14) and (15), relates to improving the rules so that they make use of morphological markers where they exist (e.g. *a* (and) will not be followed by soft mutation, whereas *a* (who, which, that) will). Other work, as in (16), will involve trying to improve number and gender agreement.

5.3. Lexical selection

Lexical selection in Apertium is under development - this would increase fluency in examples such as the following:

(17) *big / great* (adjective)
a. *Mae hyn o bwysigrwydd mawr i Gymru ac yn hanfodol i ddyheadau'r Cynulliad.*
b. *This is of big importance to Wales and essential to the aspirations of the Assembly.
c. This is of great importance to Wales and esssential to the aspirations of the Assembly.

(18) *as / like* (conjunction)

a. *Mae'r adeilad hwn yn anaddas fel cartref hirdymor i'r Cynulliad.*
b. *This building is inappropriate like long-term home to the Assembly.
c. This building is unsuitable as a long-term home for the Assembly.

6. Discussion

The collaborative development model adopted from the free and open-source software movement has a great deal to offer MT research:

1. Existing data (e.g. dictionaries, corpora) can be reused, and software can be extended. Improvements can be shared, thus ensuring that scarce resources are used to best effect, and that the materials are as robust as possible by being tested in different contexts.
2. The robustness or versatility of a research model can be tested more rigorously. The fact that Apertium can be used for language families for which it was not designed, for instance, encourages us to believe that its architecture is broadly valid.
3. Results can be verified – the concept of reproducibility is central to the scientific method, but is absent unless all of the original data and software can be executed, examined and changed as necessary.

It must be emphasised, however, that in order for MT research to describe itself as "open" or "open-source" **both** the engines **and** the data (whether rules, dictionaries, grammars, corpora or whatever) need to be available under an open licence. For marginalised languages, the latter can sometimes be a problem, in that where such data exists it may not be open, due to an immature understanding on the part of language promotion bodies of the benefits of open research – see also (Streiter, Scannell, and Stuflesser, 2006, Forcada, 2006, Koster and Gradmann, 2004). Our strong view is that any materials developed with the input of public funds should by default have at least a subset released under an open licence. During the development of *apertium-cy* we have had reason to reflect on the odd situation of being able to use materials for English developed with public money in Spain, but not materials for Welsh developed with public money in Wales.

For SMT, given a corpus for any pair of languages, it is usually possible to arrive at a working MT system after a relatively short training time. In contrast, RBMT systems have often taken years to develop, largely because of the time involved in writing the dictionaries and the rules. We would argue that for marginalised languages the collaborative development model improves the attractiveness of RBMT as an MT option, for a number of reasons:

1. Such languages may not have freely available aligned corpora available of the size required for SMT,[13] but if they have dictionaries and grammars available, writing rules based on these will be as viable as trying to generate corpora.

[13] In fact, Welsh has a large corpus (the Proceedings of the National Assembly of Wales, referred to above) that could be used for SMT, but unfortunately it is not available under a free licence.

2. Development time for open RBMT can be drastically reduced compared to closed RBMT by editing and adapting any open data that is already available in some form (e.g. dictionaries put together by enthusiasts). The authors are currently combining free Breton–Dutch and Breton–French dictionaries to create a unified 50,000-word resource, to which English can be added in due course.
3. If dictionaries and rules already exist for a series of language pairs, they can be edited and adapted to create a new pair. With Apertium, for instance, given the existing Spanish-English and Welsh-English pairs, the data and rules could be used to create a Welsh-Spanish MT system. Although the same point applies to SMT, translating a corpus may be a much more demanding affair.
4. For marginalised languages, the "critical mass" of required data or language tools may be no less than for a majority language, but the resources available (skills, time, money) are usually much less. Compared to the balkanisation of effort represented by closed systems, the decreased time (and therefore money!) required by open RBMT systems make it more feasible for such languages to get machine translation systems, and for this development to be done on a community basis.
5. Further useful tools may become available as a spin-off from the main effort – for instance, the first author was able to develop a proof-of-concept Welsh vocabulary assistant for web-pages, *Geriaoueg*, based on Apertium, in a couple of days.[14]

7. Conclusions

We have shown how a rule-based machine translation system for Welsh was quickly developed from existing data and software, and demonstrated that the translation quality in this initial version of the software is encouraging. Crucially, the materials used in the project were all available under a free licence which allowed them to be used and adapted in a collaborative development process. This also means that they are all available for review by interested researchers. We have pointed out a number of compelling reasons why the collaborative development model and rule-based MT systems are a good fit for marginalised languages.

Acknowledgements We are grateful to Liam Tomkins and Dafydd Francis for their support during the development of the *apertium-cy* system, and to Felipe Sánchez Martínez, Gema Ramírez Sánchez, Sergio Ortiz Rojas and Mikel L. Forcada for their comments and suggestions.

Bibliography

Armentano-Oller, Carme, Rafael C. Carrasco, Antonio M. Corbí-Bello, Mikel L. Forcada, Mireia Ginestí-Rosell, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Gema Ramírez-Sánchez, Felipe Sánchez-

[14] `http://elx.dlsi.ua.es/geriaoueg`

Martínez, and Miriam A. Scalco. 2006. "Open-source Portuguese-Spanish machine translation". *Proceedings of the 7th International Workshop on Computational Processing of Written and Spoken Portuguese, PROPOR-2006*.

Ball, Martin J. and Nicole Müller. 1992. *Mutation in Welsh*. Routledge.

Callison-Burch, Chris, Miles Osborne, and Philipp Koehn. 2006. "Re-evaluating the role of Bleu in machine translation research". *EACL-2006*.

Crystal, David. 2000. *Language Death*. Cambridge University Press.

Forcada, Mikel. 2006. "Open-source machine translation: an opportunity for minor languages". *Strategies for developing machine translation for minority languages (5th SALTMIL workshop on Minority Languages), LREC-2006*.

Garrido-Alenda, Alicia and Mikel L. Forcada. 2002. "Comparing nondeterministic and quasideterministic finite-state transducers built from morphological dictionaries". *Procesamiento del Lenguaje Natural (XVIII Congreso de la Sociedad Española de Procesamiento del Lenguaje Natural)*.

Jones, Dafydd and Andreas Eisele. 2006. "Phrase-based statistical machine translation between English and Welsh". *Strategies for developing machine translation for minority languages (5th SALTMIL workshop on Minority Languages), LREC-2006*.

Karlsson, F., A. Voutilainen, J. Heikkilä, and A. Anttila. 1995. *Constraint Grammar: A language independent system for parsing unrestricted text*. Mouton de Gruyter.

Koster, Cornelis H. A. and Stefan Gradmann. 2004. "The Language belongs to the People". *Proceedings of the 4th International Conference on Language Resources and Evaluation, LREC-2004*.

Labaka, Gorka, Nicholas Stroppa, Andy Way, and Kepa Sarasola. 2007. "Comparing rule-based and data-driven approaches to Spanish-to-Basque machine translation". *Proceedings of the 4th International Conference on Language Resources and Evaluation, LREC-2004*.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. "BLEU: A method for automatic evaluation of machine translation". *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

Phillips, John D. 2001. "The Bible as a basis for machine translation". *Proceedings of PACLing 2001*.

Roche, Emmanuel and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.

Scannell, Kevin P. 2008. "*An Gramadóir*: A grammar-checking framework for the Celtic languages and its applications". *14th annual NAACLT conference*.

Somers, Harold. 2004. *Machine Translation and Welsh: The Way Forward*. Welsh Language Board. Available from `http://www.byig-wlb.org.uk/english/publications/publications/2302.doc`.

Stewart, T.W. 2004. *Mutation as Morphology: Bases, Stems and Shapes in Scottish Gaelic*. Doctoral dissertation, Ohio State University.

Streiter, Oliver, Kevin P Scannell, and Mathias Stuflesser. 2006. "Implementing NLP Projects for Non-Central Languages: Instructions for Funding Bodies, Strategies for Developers". *Machine Translation*, 20(4).

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 67–78

Grammar based statistical MT on Hadoop

An end-to-end toolkit for large scale PSCFG based MT

Ashish Venugopal, Andreas Zollmann

Abstract

This paper describes the open-source Syntax Augmented Machine Translation (SAMT) [1]on Hadoop toolkit—an end-to-end grammar based machine statistical machine translation framework running on the Hadoop implementation of the MapReduce programming model. We present the underlying methodology of the SAMT approach with detailed instructions that describe how to use the toolkit to build grammar based systems for large scale translation tasks.

1. Introduction

1.1. PSCFG approaches to Machine Translation

Syntax Augmented Machine Translation (SAMT) (Zollmann and Venugopal, 2006) defines a specific parameterization of the probabilistic synchronous context-free grammar (PSCFG) approach to machine translation. PSCFG approaches take advantage of nonterminal symbols, as in monolingual parsing, to generalize beyond purely lexical translation. Consider the example rule below:

$$@\text{VP} \rightarrow \text{ne } @\text{VB}_1 \text{ pas } \# \text{ do not } @\text{VB}_1 : w$$

representing the discontiguous translation of the French words "ne" and "pas" to "do not", in the context of the labeled nonterminal symbol "@VB" (representing the syntactic constituent type of Verb). These rules seem considerably more complex than weighted word-to-word rules (Brown et al., 1993), or phrase-to-phrase rules (Koehn, Och, and Marcu, 2003, Och and Ney, 2004) but can be viewed as natural extensions to these well established approaches. An introduction to PSCFG approaches to machine translation can be found in (Chiang and Knight, 2006).

[1]Released under the GNU Lesser General Public License, version 2

Please cite this article as: Ashish Venugopal, 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 No. 91, 2009, 67–78.

(Chiang, 2005) describes a procedure to learn PSCFG rules from word-aligned parallel corpora, using the phrase-pairs from (Koehn, Och, and Marcu, 2003) as a lexical basis for the grammar. SAMT (Zollmann and Venugopal, 2006) extends the procedure from (Chiang, 2005) to assign labels to nonterminal symbols based on target language phrase structure parse trees.

In this paper, we describe an end-to-end statistical machine translation framework—SAMT on Hadoop—to learn and estimate parameters for PSCFG grammars from word-aligned parallel corpora (*training*), and perform translation (*decoding*) with these grammars under a log-linear translation model (Och and Ney, 2004). While our framework specifically implements (Chiang, 2005) and (Zollmann and Venugopal, 2006), the training and decoding algorithms in our toolkit can be easily replaced to experiment with alternative PSCFG parameterizations like (Galley et al., 2006, Wu, 1997). The algorithms in this toolkit are implemented upon Hadoop (Cutting and Baldeschwieler, 2007), an open-source implementation of the MapReduce (Dean and Ghemawat, 2004) framework, which supports distribution computation on large scale data using clusters of commodity hardware. We report empirical results that demonstrate the use of the SAMT toolkit on large scale translation tasks.

1.2. The SAMT toolkit

Our toolkit, when used in concert with other open-source components and publicly available corpora, contains all of the necessary components to build and evaluate grammar based statistical machine translations systems. The primary components of the toolkit are listed below:

- A top level push-button script that provides experimental work-flow management and submits jobs to the underlying Hadoop framework.
- Components to build and estimate parameters for the grammars described in (Chiang, 2005) and (Zollmann and Venugopal, 2006).
- Tools to filter large translation grammars and n-gram language models to build small sentence specific models that can be easily loaded into memory during decoding.
- A bottom-up dynamic chart parsing decoder based on (Chappelier and Rajman, 1998) which supports grammars with more than 2 nonterminals symbols per rule. The decoder outputs n-best lists with optional annotations that facilitate discriminative training.
- An implementation of Minimum Error Rate (MER) training (Och, 2003), extended to perform feature selection.

The SAMT toolkit requires the following inputs that are easily generated by existing open-source tools.

- Word aligned parallel corpora. For small resource tasks, word-alignments can be generated using the GIZA++ toolkit (Och and Ney, 2003), while large-resource tasks can be aligned using (Dyer et al., 2008), a parallelized GIZA++ implementation on MapReduce.
- (Zollmann and Venugopal, 2006) requires target language parse trees for each sentence in the training data. SAMT on Hadoop interfaces to the parser from (Charniak, 2000) to parse the target side of the parallel corpora on Hadoop.
- N-Gram language models built via the SRILM toolkit (Stolcke, 2002) are used as features

during decoding.

1.3. SAMT on Hadoop

The SAMT toolkit is built upon Hadoop (Cutting and Baldeschwieler, 2007), an open-source implementation of the MapReduce model to distribute the estimation of PSCFG grammars and to perform decoding. Training and decoding are broken up into a series of MapReduce tasks, called *phases*, which are performed sequentially, transforming input data into a PSCFG grammar, and using the grammar to translate development and test sentences. Phase outputs are stored on the Hadoop Distributed File System (HDFS), a highly fault tolerant file system that is accessible by all cluster machines. Most SAMT phases are run sequentially, using output from previous phases as input[2]. Detailed instructions for downloading and building the SAMT toolkit are available at the toolkit's website[3], along with examples that that can be used to re-generate published results from (Zollmann, Venugopal, and Vogel, 2008). In the remainder of this paper, we describe the SAMT methodology and important user parameters in our toolkit that impact translation quality and runtime. For a more formal description of the individual MapReduce phases in the SAMT pipeline, see (Zollmann, Venugopal, and Vogel, 2008).

2. Syntax Augmented Machine Translation

2.1. Phrase and SAMT Rule Extraction

In this section, we describe Syntax Augmented Machine Translation (SAMT) (Zollmann and Venugopal, 2006), a specific instantiation of the PSCFG formalism that is implemented in the SAMT on Hadoop toolkit. SAMT extends the purely hierarchical grammar proposed in (Chiang, 2005) to use nonterminal labels learned from target language parse trees. The inputs to the SAMT rule extraction procedure are tuples, $\langle f, e, \mathtt{Phrases}(a, f, e), \pi \rangle$, where f is a source sentence, e is a target sentence, a is a word-to-word alignment associating words in f with words in e, $\mathtt{Phrases}(a, e, f)$, are the set of phrase pairs (source and target phrases) consistent with the alignment a (Koehn, Och, and Marcu, 2003, Och and Ney, 2004), and π is a phrase structure parse tree of e. SAMT rule extraction associates each phrase pair from $\mathtt{Phrases}(a, e, f)$ with a left-hand-side label, and then applies the rule extraction procedure from (Chiang, 2005) to generate rules with *labeled* nonterminal symbols.

Consider the example alignment graph (a word alignment and target language parse tree as defined in (Galley et al., 2006)) for the example French-to-English sentence in Figure 1. The phrase extraction method from (Koehn, Och, and Marcu, 2003), extracts all phrase pairs where no word inside the phrase pair is aligned to a word outside the phrase pair. Figure 2 gives the initial rules extracted for our example sentence pair.

[2]While these scripts assume the Hadoop-on-Demand machine requisitioning model, the toolkit can be easily modified to submit jobs to a single global machine pool

[3]www.cs.cmu.edu/~zollmann/samt

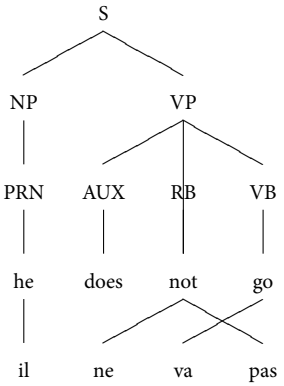

Figure 1. Alignment graph (word alignment and target parse tree) for a French-English sentence pair.

PRP:NP	→	il # he
VB	→	va # go
RB+VB	→	ne va pas # not go
VP	→	ne va pas # does not go
S	→	il ne va pas # he does not go

Figure 2. Labeled initial rules.

S	→	PRP:NP$_1$ ne va pas # PRP:NP$_1$ does not go
S	→	il ne VB$_1$ pas # he does not VB$_1$
S	→	il VP$_1$ # he VP$_1$
S	→	il RB+VB$_1$ # he does RB+VB$_1$
S	→	PRP:NP$_1$ VP$_2$ # PRP:NP$_1$ VP$_2$
S	→	PRP:NP$_1$ RB+VB$_2$ # PRP:NP$_1$ does RB+VB$_2$
VP	→	ne VB$_1$ pas # does not VB$_1$
RB+VB	→	ne VB$_1$ pas # not VB$_1$
VP	→	RB+VB$_1$ # does RB+VB$_1$

Figure 3. Generalized rules.

Phrase Extraction is the first phase of the SAMT toolkit, annotating each sentence-pair of the training corpus with a set of phrase pairs extracted from that sentence pair. We use a single toolkit binary: **MapExtractPhrases**, run as Hadoop Map step (there is no Reduce step in this phase). This binary takes a single numberical argument which determines the maximum length of the initial phrase extracted from word-aligned data. This limit has an impact on the size and nature of the final grammar. Typically, phrase limits are significantly smaller than the length of the parallel sentence, preventing very long distance reordering effects from being captured in the grammar.

The next phase, Rule Extraction includes rule indetification (Map step, binary **MapExtractRules**) on a per-sentence basis, and merging and counting of identical rules (Reduce step, binary **MergeRules**). SAMT assigns a left-hand-side (lhs) label to every phrase pair extracted from the current sentence-pair, based on the corresponding target language parse tree π, forming *initial rules*. These labels are assigned based on the constituent spanning the target side word sequence in π. When the target side of the phrase-pair is spanned by a single constituent in π, the constituent label is assigned as the lhs for the phrase pair. If the target side of the phrase is not spanned by a single constituent in π, we use the labels of subsuming, subsumed, and neighboring constituents in π to assign an extended label of the form $C_1 + C_2$, C_1/C_2, or $C_2\backslash C_1$ (similar in motivation to the labels in (Steedman, 1999)), indicating that the phrase pair's target side spans two adjacent syntactic categories (e.g., *she went: NP+VB*), a partial syn-

tactic category C_1 missing a C_2 at the right (e.g., *the great: NP/NN*), or a partial C_1 missing a C_2 at the left (e.g., *great wall: DT\NP*), respectively. The label assignment is attempted in the order just described, i.e., assembling labels based on '+' concatenation of two subsumed constituents is preferred, as smaller constituents tend to be more accurately labeled. If no label is assignable by either of these three methods, and the parameter *'-allow_double_plus 1'* is set, we try triple-concatenation to create a label of the form $C_1 + C_2 + C_3$. If this approach do not yield a label or if *'-allow_double_plus 0'*, a default label '_FAIL' is assigned. An ambiguity arises when unary rules $N_1 \rightarrow ... \rightarrow N_m$ in the target parse tree are encountered, such as the NP→PRN subtree in Figure 1. Depending on the parameter *'-unary_category_handling'*, we use the bottom-most label (parameter value *'bottom'*), the top-most (*'top'*), or a combined label $N_m : \ldots : N_1$ (*'all'*, this is the default).

An alternative method of assigning labels to phrase pairs can be activated by specifying the parameter *'-use_only_pos'*. In this variant, labeling is performed merely based on the part-of-speech (POS) tags of the first word *POS1* and last word *POS2* of the target phrase, resulting in the label *'POS1-POS2'*. In general, the SAMT approach can take advantage of any labeling techniques that assigns labels to arbitrary initial phrase pairs. Alternative techniques could include using source language constituent labels, or automatically induced labels. Based on these initial rules, we perform the rule *generalization* procedure from (Chiang, 2005). Figure 3 shows the resulting generalized rules. For each labeled rule in the grammar, we can also generate a corresponding generically labeled rule as in (Chiang, 2005). We introduce an additional feature in the log-linear translation model that allows the decoder to prefer labeled or unlabeled derivations. To suppress the creation of generic rules, pass the parameter *'-generate_generic_variant 0'*.

The number of rules generated by this procedure is exponential in the number of initial phrases pairs, producing a grammar that is impractical for efficient translation. The following parameters are used to restrict the number of rules extracted per sentence:

- *-max_abstraction_count* (default: 2): maximum number of abstractions (nonterminal pairs) per rule.
- *-max_source_symbol_count* (default: 6): maximum number of symbols (terminals and nonterminals) on the source side of the rule.

This restricted rule set can be pruned further with the following parameters for **MergeRules**:

- *-allow_consec_nts* (default: 1): if set to 0, discards rules that have consecutive nonterminals on the source side.
- *-allow_src_abstract* (default: 1): if 0, discards rules that do not have any source terminal symbols for example: $S \rightarrow NP\ _1VP_2\ \#\ NP\ _2VP_1$. Setting this parameter to 0, drastically reduces decoding time.
- *-nonlexminfreq, -lexminfreq* (defaults: 0): minimum occurrence frequency thresholds for non-lexical and lexical rules respectively. Increasing these thresholds reduces the size of the grammar, but often at the cost of translation quality (Zollmann et al., 2008).
- *-min_freq_given_src_arg* (default: 0): minimum relative frequency of a rule given its labeled source.

The labeling and extraction procedures defined above identify rules from the input word-

aligned parallel corpora and associated parse trees. The occurrence counts from this extraction process are used in estimating relative frequency features for each rule. The estimation of these features is described in the next section.

2.2. PSCFG Features

Given a source sentence f and a PSCFG grammar, the translation task can be expressed analogously to monolingual parsing with a CFG. We find the most likely derivation D of the input source sentence and read off the English translation, identified by composing α from each rule used in the derivation. This search for the most likely derivation can be defined as:

$$\hat{e} = \text{tgt}\left(\underset{D \in \text{Derive}(G):\text{src}(D)=f}{\arg\max} p(D)\right) \tag{1}$$

where $\text{tgt}(D)$ refers to the sequence of target terminal symbols generated by the derivation D, $\text{src}(D)$ refers to the source terminal symbols of D and $\text{Derive}(G)$ is the set of sentence spanning derivations of grammar G. The distribution p over derivations is defined by a log-linear model. The probability of a derivation D is defined in terms of the rules r that are used in D:

$$p(D) = \frac{p_{LM}(\text{tgt}(D))^{\theta_{LM}} \prod_{r \in D} \prod_{i=1}^{m} \lambda_i(r)^{\theta_i}}{Z(\theta_{LM}, \theta_1, \ldots, \theta_m)} \tag{2}$$

where $\lambda_i(r)$ refers to features defined on each rule, p_{LM} is an n-gram language model (LM) probability distribution over target word sequences, and Z is a normalization constant that does not need to be computed during search under the arg max search criterion in Equation 1. The feature weights $\theta_{LM}, \theta_1, \ldots, \theta_m$ are trained in concert with the language model weight via MER training. The features $\lambda_i(r)$ are statistics estimated from rule occurrence counts.

The output of the Rule Extraction phase is a grammar with a small subset of features in λ that has been learned automatically from the input data. The features used in the our toolkit include those in (Chiang, 2005, Zollmann and Venugopal, 2006), and are computed in the Rule Extraction and Filtering phase (described below). The resulting grammar is large, and for most translations tasks, cannot be loaded directly into memory for decoding. To avoid this problem, the SAMT toolkit filters the grammar against a specific test corpus, generating a sentence specific grammar for each *sentence* in the corpus. This filtering is performed for each corpora that we need for translation, typically *development*, *test*, and *unseen test* corpora are used to train and evaluate machine translation systems.

2.3. Rule and LM Filtering

The Rule Filtering phase (binaries **MapSubsampleRules**, **filterrules_bin**) take as input: the grammar from the Rule Extraction phase, a corpus to filter the grammar against, and additional model files (such as translation lexica) to generate additional rule features in λ. In the

Map step, the grammar is filtered on a per-sentence basis by matching the source words of each rule to the source words in the sentence we want to translate. In the Reduce step, additional features (documented in **filterrules.pl**, which is used to generate the MapReduce binary **filterrules_bin**). The Reduce step of Rule Filtering provides several options to further restrict the grammar and to augment the additional features. These options can be specified via the top-level parameter: *filter_params*. The Rule Filtering Reduce step also adds the following system rules to each sentence specific grammar.

- Beginning-of-sentence rule: $S \rightarrow \langle s \rangle \ \# \ \langle s \rangle$
- Glue rules (Chiang, 2005) for each NT N in the grammar, for example: $S \rightarrow S_1 N_2 \ \# \ S_1 N_2$
- End-of-sentence rule: $S \rightarrow S_1 \langle \backslash s \rangle \ \# \ S_1 \langle \backslash s \rangle$
- 'Unknown'-rules (e.g. NNP→ _UNKNOWN # _UNKNOWN) generating a limited set of labels for the word '_UNKNOWN', which the decoder substitutes for unknown source words

The Glue rules (Chiang, 2005) play an important role in grammar based approaches to MT. These rules serve to simply concatenate translations of consecutive spans during decoding, similar to monotone decoding in a phrase based system (Koehn, Och, and Marcu, 2003). These Glue operations allow the system to produce translations that violate the syntactic contraints encoded in the labels of the grammar—at a cost determined via the MER trained weight $\theta_{\texttt{glue}}$.

Building sentence specific grammars allows us to estimate the parameters and features of the grammar on large parallel corpora, while still being able to load all relevant rules to translate particular sentences in a test corpus. We follow this same approach to filter large n-gram language models in a LM Filtering phase. While the Rule Filtering phase filters rules based on the source side of the rule, the n-gram LM must be filtered according to the possible set of target words that can generated by applying the sentence specific grammar. For each sentence specific grammar, a possible target vocabulary is generated, which is used by the Rule Filtering binary (**LMFilter**) to produce sentence specific language models.

3. PSCFG Decoding

The runtime complexity of our decoder with an integrated n-gram LM feature is:

$$\mathcal{O}\left(|f|^3 \left[|\mathcal{N}||\mathcal{T}_T|^{2(n-1)}\right]^K\right) \tag{3}$$

where K is the maximum number of NT symbols per rule, $|f|$ is the source sentence length, $\mathcal{N}$ is the set of nonterminal labels in the grammar, $\mathcal{T}_T$ is the set of target language terminals the in grammar, and n is the order of the n-gram LM. Our decoder implements the Cube Pruning algorithm from (Chiang, 2007), and outputs n-best lists for use in MER. The **FastTranslateChart** performs translation as a Map task. The grammar restriction parameters described in Section 2.1 have a large significant impact on decoding runtime (particulary *allow_src_abstract, allow_consec_nts, max_abstraction_count*), but this search still requires additional pruning to

produce translations in reasonable time-frames—especially when translating longer sentences. The most important decoder parameters are described below:

- *wts*: corresponds to the weights θ in the translation model in Equation 2. In practice, these weights are iteratively trained via MER.
- *HistoryLength*: (default 2) The number of words considered as LM history length during decoding. When set to less than $n - 1$, when using an n-gram LM, decoding time is reduced at the expense of search errors, which can reduce translation quality.
- *SRIHistoryLength*: This value indicates the full history length of the n-gram language model. When using a reduced *HistoryLength*, this value is used to recover from search errors in a LM-driven n-best extraction step similar to (Huang and Chiang, 2007).
- *PruningMap*: (default: 0-100-5-@_S-200-5): Format: lhs-b-β. Pruning parameters for Cube Pruning (Chiang, 2007). For each nonterminal label lhs in the grammar for a source span during decoding, this parameter restricts the number of chart items to b items, and items that are have cost of at most β greater than the best item. lhs $= 0$ sets pruning parameters for all lhs symbols that have not been explicly specified.
- *ComboPruningBeamSize* : (default 10000) Sets the maximum number of items generated in each cell via Cube Pruning. Reducing this value reduces decoding time when *PruningMap* limits have not caused pruning.
- *MaxHypsPerCell*: (default 1000000000) Limits the total number of items (partial translation hypotheses) created for each span during decoding—across items that have different lhs labels (not counting X and S items, which always pass thru this pruning filter). This value is typically set when using grammars with a large number of lhs labels to reduce translation runtime, but does introduce additional search error.
- *MaxCostDifferencePerCell*: (default inf) Max. allowed cost that an item can deviate from the best item in its chart cell (inf: any cost allowed). Items with lhs X or S always pass thru this filter. This and the previous parameter are the only paramters that apply pruning across items with different nonterminal labels.
- *MaxCombinationCount*: (default 10) Limits the application of automatically learned PSCFG rules to source spans less than or equal to *MaxCombinationCount*. Spans of greater length are composed monotonically with Glue rules. Decoding time is linear in sentence length once this limit is in effect.

3.1. Minimum Error Rate Training

The parameters θ are trained via MER training to maximize translation quality according to a user specified automatic translation metric, like BLEU (Papineni et al., 2002) or NIST (Doddington, 2002). MER training is implemented in the SAMT toolkit as a MapReduce phase using n-best lists from the decoding phase. Our MER implementation performs feature selection, preferring solutions where $\theta_i = 0$, and can be easily extended to perform random restarts as well.

Track	Words (English)	LM 1-N grams (N)	Dev.	Test1	Test2
IWSLT	632K	431,292 (5)	IWSLT06	IWSLT07	N/A
67M	67M	102,924,025 (4)	MT05	MT06	MT08
230M	230M	273,233,010 (5)	MT05	MT06	MT08

Table 1. Training data configurations used to evaluated SAMT on Hadoop. The number of words in the target text and the number of 1-N grams represented in the complete model are the defining statistics that characterize the scale of each task. For each LM we also indicate the order of the n-gram model.

System	Dev. BLEU	Test1 BLEU	Test2 BLEU	Grammar Train. (h:m)	Test1 (m)
IWSLT Hier	27.0	37.0	N/A	0:12	4
IWSLT Syntax	30.9	37.2	N/A	0:26	12
67M Hier	35.19	32.98	25.88	1:10	17
67M Syntax	35.69	33.12	26.48	2:26	65
230M Hier	36.39	33.74	26.28	4:13	23
230M Syntax	37.11	34.04	26.74	7:21	53

Table 2. Translation quality as measured by IBM-BLEU% (i.e., brevity penalty based on closest reference length) on each resource track for appropriate evaluation data sets. Systems 67M and 230M are evaluated in lower-case, while IWSLT is evaluated in mixed case. Training and decoding times given are based on a cluster of 100 (2 per machine) 1.9GHz Intel Xeon processors.

4. Empirical Results

We demonstrate the SAMT on Hadoop toolkit on three Chinese-to-English translation tasks, representing a wide range of resource conditions. Each task is described in Table 1. The IWSLT task is a limited resource, limited domain task, while 67M and 230M (named for their respective corpora sizes), are corpora used for the annual NIST MT evaluation. For each task we list the number of words in the target side of the corpus and the number of 1-n grams in the n-gram LM (estimated from parallel and monolingual data).

For each resource condition, we build SAMT systems using a purely hierarchical grammar (Hier) (Chiang, 2005) and a syntax augmented grammar (Syntax) from (Zollmann and Venugopal, 2006). All experiments use a 2-gram *HistoryLength* length the first pass of decoding, and the full LM history during the second pass n-best list search. These grammars are built with *'-allow_consec_nts 0 -allow_src_abstract 0'*, and the NIST MT task rules are additionally restricted by *'-nonlexminfreq 2 -min_freq_given_src_arg* α*'* where $\alpha = 0.005$ (Hier) and $\alpha = 0.01$ (Syntax). The Syntax based systems also use *'-MaxHypsPerCell 1000'* to limit the run time impact of the large number of lhs labels in these grammars.

In Table 2, we report BLEU scores on development and test data as well as run times to train

the respective PSCFG grammars and perform translation with them. Training run times are reported based on Hadoop MapReduce jobs running on a cluster of 50 dedicated machines, each running 2 Map or Reduce tasks each. These results demonstrate the ability for the SAMT toolkit to scale to large resource data conditions. For each of the the three data conditions we see that training the Syntax grammar takes longer to train as well as translate with. Translation quality improvements that result from using more parallel and monolingual data are clear when comparing the 67M and 230M systems. In these experiments, we see small but consistent improvements from the introduction of SAMT labels, in line with experiments in (Zollmann et al., 2008). Overall, translation quality results reported here are competitive with reported results in the literature and constitute a valid baseline for further research.

5. Conclusions and Resources

In this paper we have described the SAMT on Hadoop toolkit, an end-to-end framework for large scale grammar based statistical machine translation. We discussed the methodology of the SAMT approach, and described important toolkit parameters that affect translation quality and run time. Built upon the open-source Hadoop distributed computation framework, our toolkit is able to scale to build grammars for large scale translation tasks in reasonable time frames. The toolkit can be easily extended to experiment with alternative grammar extraction and decoding techniques.

Bibliography

Brown, Peter F., Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*.

Chappelier, J.C. and M. Rajman. 1998. A generalized CYK algorithm for parsing stochastic CFG. In *Proceedings of Tabulation in Parsing and Deduction (TAPD)*, pages 133–137, Paris.

Charniak, Eugene. 2000. A maximum entropy-inspired parser. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics Conference (HLT/NAACL)*.

Chiang, David. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Compuational Linguistics (ACL)*.

Chiang, David. 2007. Hierarchical phrase based translation. *Computational Linguistics*.

Chiang, David and Kevin Knight. 2006. An introduction to synchronous grammars. In *Tutorials at the Annual Meeting of the Association for Compuational Linguistics (ACL)*.

Cutting, Doug and Eric Baldeschwieler. 2007. Meet Hadoop. In *O'Reilly Open Software Convention*, Portland, OR.

Dean, Jeffrey and Sanjay Ghemawat. 2004. Mapreduce: Simplified data process on large cluster. In *Proceedings of Symposium on Operating System Design and Implementation*.

Doddington, George. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *In Proceedings ARPA Workshop on Human Language Technology*.

Dyer, Christopher, Aaron Cordova, Alex Mont, and Jimmy Lin. 2008. Fast, easy, and cheap: Construction of statistical machine translation models with mapreduce. In *Proceedings of the Workshop on Statistical Machine Translation, ACL*.

Galley, Michael, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2006. Scalable inferences and training of context-rich syntax translation models. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics Conference (HLT/NAACL)*.

Huang, Liang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the Annual Meeting of the Association for Compuational Linguistics (ACL)*.

Koehn, Philipp, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics Conference (HLT/NAACL)*.

Och, Franz J. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Compuational Linguistics (ACL)*.

Och, Franz J. and Hermann Ney. 2003. A systematic comparison of various alignment models. *Computational Linguistics*.

Och, Franz J. and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting of the Association for Compuational Linguistics (ACL)*.

Steedman, Mark. 1999. Alternative quantifier scope in CCG. In *Proceedings of the Annual Meeting of the Association for Compuational Linguistics (ACL)*.

Stolcke, Andreas. 2002. SRILM —an extensible language modeling toolkit. In *Proceedings of the International Conferrence on Spoken Language Processing (ICSLP)*.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*.

Zollmann, Andreas and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation, HLT/NAACL*, New York, June.

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

Zollmann, Andreas, Ashish Venugopal, and Stephan Vogel. 2008. The CMU Syntax-Augmented Machine Translation System: SAMT on Hadoop with N-best Alignments. In *Proc. of the International Workshop on Spoken Language Translation*, pages 18–25, Hawaii, USA.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 79–88

Z-MERT: A Fully Configurable Open Source Tool for Minimum Error Rate Training of Machine Translation Systems

Omar F. Zaidan

Abstract

We introduce **Z-MERT**, a software tool for minimum error rate training of machine translation systems (Och, 2003). In addition to being an open source tool that is extremely easy to compile and run, Z-MERT is also agnostic regarding the evaluation metric, fully configurable, and *requires no modification to work with any decoder*. We describe Z-MERT and review its features, and report the results of a series of experiments that examine the tool's runtime. We establish that Z-MERT is extremely efficient, making it well-suited for time-sensitive pipelines. The experiments also provide an insight into the tool's runtime in terms of several variables (size of the development set, size of produced N-best lists, etc).

1. Introduction

Many state-of-the-art machine translation (MT) systems over the past few years (Och and Ney, 2002, Koehn, Och, and Marcu, 2003, Chiang, 2007, Koehn et al., 2007) rely on several models to evaluate the "goodness" of a given candidate translation in the target language. The MT system proceeds by searching for the highest-scoring candidate translation, as scored by the different model components, and returns that candidate as the hypothesis translation. Each of these models need not be a probabilistic model, and instead corresponds to a feature that is a function of a (candidate translation,foreign sentence) pair.

Treated as a log-linear model, we need to assign a weight for each of the features. Och (2003) provides empirical evidence that setting those weights should take into account the evaluation metric by which the MT system will eventually be judged. This is achieved by choosing the weights so as to maximize the performance of the MT system on a development set, as measured by that evaluation metric. The other insight of Och's work is that there exists an efficient algorithm to find such weights.

This process has come to be known as the MERT phase (for Minimum Error Rate Training) in training pipelines of MT systems. The existence of a MERT module that can be integrated

Please cite this article as: Omar F. Zaidan, Z-MERT: A Fully Configurable Open Source Tool for Minimum Error Rate Training of Machine Translation Systems. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 79–88.

with minimal effort with an existing MT system would be beneficial for the research community. For maximum benefit, this tool should be easy to set up and use and should have a demonstrably efficient implementation. We describe here one such tool, **Z-MERT**, developed with these goals in mind. Great care has been taken to ensure that *Z-MERT can be used with any MT system without modification to the code*, and without the need for an elaborate[1] web of scripts, which is a situation that unfortunately exists in practice in current training pipelines.

We first review log-linear models in MT systems and Och's efficient method (Section 2) before introducing Z-MERT and its usage (Section 3). We also report experimental results that demonstrate Z-MERT's efficiency (Section 4). Finally, we provide details on how to take full advantage of Z-MERT's unique features (Section 5). Readers already familiar with MERT should feel free to skip to the last paragraph of Section 2.

2. Log-linear Models in Machine Translation

Given a sentence to translate f in the source (aka 'foreign') language, a MT system attempts to produce a hypothesis translation $\hat{e}$ in the target (aka 'English') language that it believes is the best translation candidate. This is done by choosing the target sentence with the highest probability conditioned on the given source sentence. That is, the chosen translation is:

$$\hat{e} = \underset{e}{\operatorname{argmax}} \Pr(e \mid f) \tag{1}$$

One could model the posterior probability $\Pr(e \mid f)$ using a *log-linear model*. Such a model associates a sentence pair (e, f) with a feature vector $\Phi(e, f) = \{\phi_1(e, f), ..., \phi_M(e, f)\}$, and assigns a score

$$s_\Lambda(e, f) \stackrel{\text{def}}{=} \Lambda \cdot \Phi(e, f) = \sum_{m=1}^{M} \lambda_m \phi_m(e, f) \tag{2}$$

for that sentence pair, where $\Lambda = \{\lambda_1, ..., \lambda_M\}$ is the weight vector for the M features. Now, the posterior is defined as:

$$\Pr(e \mid f) \stackrel{\text{def}}{=} \frac{\exp(s_\Lambda(e, f))}{\sum_{e'} \exp(s_\Lambda(e', f))} \tag{3}$$

and therefore, the MT system selects the translation:

$$\hat{e} = \underset{e}{\operatorname{argmax}} \Pr(e \mid f) = \underset{e}{\operatorname{argmax}} \frac{\exp(s_\Lambda(e, f))}{\sum_{e'} \exp(s_\Lambda(e', f))} = \underset{e}{\operatorname{argmax}}\, s_\Lambda(e, f). \tag{4}$$

2.1. Parameter Estimation Using Och's Method

How should one set the weight vector Λ? Och (2003) argues it should be chosen so as to maximize the system's performance on some development dataset as measured by the evaluation metric of interest. The error surface in this approach is not smooth, which means that

[1] *Elaborate*, as in complicated, hard to navigate, and headache-inducing.

gradient-based optimization techniques cannot be used. A grid search by repeated line optimizations is not a good option either, since the function is quite expensive to evaluate at a given point $p \in \Re^M$, as this would require rescoring the candidate set[2] for each sentence to find the 1-best translations at p. Och suggests an alternative efficient approach for this optimization, which we review here.

Assume we are performing a line optimization along the d^{th} dimension. That is, we have a weight vector $\Lambda = \{\lambda_1, ..., \lambda_d, ..., \lambda_M\}$, and we would like to find a new weight vector for which the d^{th} dimension is optimal, keeping the other dimensions fixed.

Consider a foreign sentence f, and let the candidate set[2] for f be $\{e_1, ..., e_K\}$. Recall from (4) that the 1-best candidate at a given Λ is the one with maximum $s_\Lambda(e_k, f)$, which (2) defines as $\sum_{m=1}^{M} \lambda_m \phi_m(e_k, f)$. We can rewrite that sum as $\lambda_d \phi_d(e_k, f) + \sum_{m \neq d} \lambda_m \phi_m(e_k, f)$. The second term is constant with respect to λ_d, and so is $\phi_d(e_k, f)$. If we rename those two quantities $\texttt{offest}_\Lambda(e_k)$ and $\texttt{slope}(e_k)$:

$$s_\Lambda(e_k, f) = \texttt{slope}(e_k)\lambda_d + \texttt{offset}_\Lambda(e_k). \tag{5}$$

This is the equation for a line, and so when we vary λ_d, the score of a candidate varies linearly. That is, if we plot the score for a candidate translation vs. λ_d, that candidate will be represented by a line. If we plot the lines for all candidates (Figure 1), then the upper envelope of these lines indicates the best candidate at any value for λ_d. This is basically a visualization of the decision process of (4).

Observe now that the non-smoothness of the error function surface is not arbitrary, but is in fact *piece-wise linear* along the λ_d dimension.[3] The reason is that the error is calculated based on the 1-best candidate translations, and a small change in λ_d usually does not change the top candidate. There is, however, a set of critical values along the λ_d dimension, corresponding to the intersection points that form the abovementioned upper envelope. These are the only points at which the error changes (due to a change in the set of 1-best candidates).

If we can determine these intersection points for each sentence, and then merge them all into one set of intersection points, we will then have an overall set of critical values along the λ_d dimension, with each value corresponding to a 1-best change for a single foreign sentence.[4]

This means that if we have already calculated the error's sufficient statistics for a λ_d value just before some critical value, the sufficient statistics for a λ_d value just after that critical value can be calculated quite easily: simply adjust the original sufficient statistics as dictated by the candidate change associated with that intersection point.

This way, there is no need to rescore the candidates, and we traverse the λ_d dimension by considering only intersection points to find the optimum value. Finding those critical values amounts to finding intersection points of the lines representing the candidates, which is an easy

[2]We have not yet indicated how this candidate set is obtained, but will do so shortly.

[3]Or, in fact, along any linear combination of the M dimensions.

[4]In theory, a single critical value might correspond to a 1-best change for more than one foreign sentence. Though infrequent, this does happen in practice and is accounted for in Z-MERT.

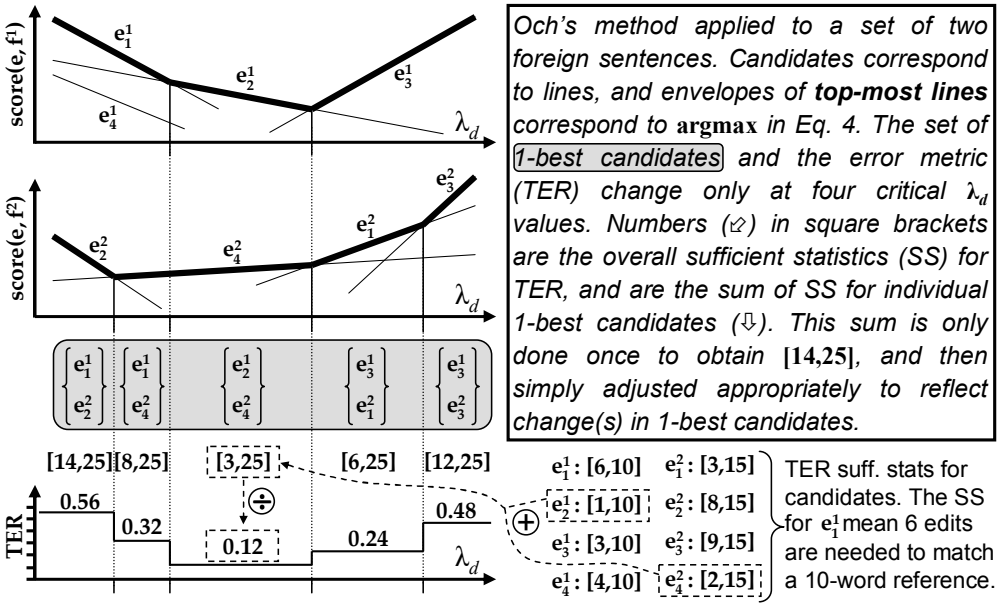

Figure 1. Och's method applied to a set of two foreign sentences.

process.[5]

One final piece of the puzzle is needed. We have been assuming that we have access to the set of candidate translations for each foreign sentence. How is this set actually obtained? One could try to enumerate *all* the possible candidates to cover the entire search space, but that may not be possible, and is likely quite costly anyway.

So we need an approximation to the candidate set. We could use the top, say, 300 candidates according the initial weight vector, but this set is quite concentrated, and is therefore not a good representative of the search space.[6] Instead, we alternate between optimizing the weight vector and producing the set of top candidates, each time merging the new candidate set with the existing candidates. The process is repeated until convergence, indicated by the candidate set not growing in size.

Och's method corresponds to line 17 in Algorithm 1, which is the pseudocode for Z-MERT's optimization process. Notice that Z-MERT repeatedly performs a line optimization (lines 14–27) along one of the M dimensions, greedily selecting the one that gives the most gain (lines 16–23). Notice also that each iteration optimizes several random "initial" points (line 9) in addition to the one surviving from the previous MERT iteration (line 8). This is used as an alternative to true multiple restarts.

[5]And it can be done efficiently: many of the lines need not be considered at all, such as the one for e_4^1 in Figure 1.

[6]We are not referring to the small **number** of candidates here (which we already accept as a compromise to avoid

Algorithm 1 Z-MERT: Optimization of weight vector Λ to minimize error, using for line optimization (line 17) the efficient method of Och (2003).

Input: Initial weight vector $\Lambda^0 = \{\Lambda^0[1], ..., \Lambda^0[M]\}$; `numInter`, the number of initial points per iteration; and `N`, the size of the candidate list generated each iteration.
Return: Final weight vector $\Lambda^* = \{\Lambda^*[1], ..., \Lambda^*[M]\}$.

1. Initialize $\Lambda \leftarrow \Lambda^0$
2. Initialize `currError` $\leftarrow +\infty$
3. Intialize the cumulative candidate set for each sentence to the empty set.
4. **loop**
5. Using Λ, produce an `N`-best candidate list for each sentence, and merge it with the cumulative candidate set for that sentence.
6. **if** no candidate set grew **then** Return Λ // MERT convergence; we are done.
7.
8. Initialize $\Lambda_1 \leftarrow \Lambda$
9. **for** ($j = 2$ to `numInter`), initialize $\Lambda_j \leftarrow$ random weight vector
10.
11. Initialize $j_{best} \leftarrow 0$
12. **for** ($j = 1$ to `numInter`) **do**
13. Initialize $currError_j \leftarrow error(\Lambda_j)$ based on cumulative candidate sets
14. **repeat**
15. Initialize $m_{best} \leftarrow 0$
16. **for** ($m = 1$ to M) **do**
17. Set (λ,`err`) = value returned by efficient investigation of the m^{th} dimension and the error at that value (i.e. using Och's method)
18. **if** ($err < currError_j$) **then**
19. $m_{best} \leftarrow m$
20. $\lambda_{best} \leftarrow \lambda$
21. $currError_j \leftarrow err$
22. **end if**
23. **end for**
24. **if** ($m_{best} \neq 0$) **then**
25. Change $\Lambda_j[m_{best}]$ to λ_{best}
26. **end if**
27. **until** ($m_{best} == 0$)
28. **if** ($currError_j < currError$) **then**
29. $currError \leftarrow currError_j$
30. $j_{best} \leftarrow j$
31. $\Lambda \leftarrow \Lambda_j$
32. **end if**
33. **end for**
34. **if** ($j_{best} == 0$) **then** Return Λ // Could not improve any further; we are done.
35. **end loop**

3. Z-MERT

Z-MERT is part of a larger effort at Johns Hopkins University to develop **Joshua** (Li and Khudanpur, 2008) into an open source software package that includes a hierarchical phrase-based decoder (Chiang, 2007), as well as the components of a complete MT training pipeline. Two principles established by the developers were flexibility and ease of use, and were observed in the development of Z-MERT, Joshua's MERT module. Z-MERT also functions independently as a standalone application. That is, Z-MERT is also publicly available[7] separately from Joshua, which is still under development, since Z-MERT does not rely on any of Joshua's other components.

3.1. Existing MERT Implementations

At first, it seemed reasonable to use some existing MERT implementation as a starting point for Joshua's MERT module and adapt it as we see fit. (After all, there is no point in reinventing the wheel.) However, we found that existing implementations were not suitable for our needs, and did not meet our standards of flexibility and ease of use. We review here two such open source MERT implementations.

The first MERT implementation we examined is by Ashish Venugopal[8], which appears to have been first used by Venugopal and Vogel (2005). One immediate drawback of this implementation is that it is written in MATLAB®, which, like other interpreted languages, is quite slow.[9] Furthermore, MATLAB® is a proprietary product of The MathWorks, which limits use of Venugopal's implementation to those who have access to a licensed installation of MATLAB®.

Beyond that, the tool needs to be launched after every decoding step to perform the MERT optimization.[10] The user could certainly opt out of monitoring the MERT process to manually launch the decoder at the end of each MERT run (and vice versa) by writing a script capable of monitoring the two processes and launching them at appropriate times. But writing such a script seems like an unnecessary nuisance.

Z-MERT, on the other hand, is written in Java, making it orders of magnitude faster. This also makes it usable by practically everybody, since Java compilers are freely available for all common platforms, and users are likely to already have one installed and be familiar with it. Z-MERT also requires no monitoring from the user – all the user needs to do is specify the command that launches the decoder, and Z-MERT takes care of everything else.

---

enumerating all the candidates), but their limited **distribution**.

[7] Software and documentation available at: http://www.cs.jhu.edu/~ozaidan/zmert.html.

[8] Software and documentation available at: http://www.cs.cmu.edu/~ashishv/mer.html.

[9] It should be noted that the sufficient statistics for error are calculated outside MATLAB®, since it is a "costly process which is not well suited to MATLAB," according to the documentation. This implies an external script in some other language performs those calculations. It is not clear *which* language, since those scripts do not appear to be available for download on the software's page.

[10] It essentially performs a single iteration of the outermost loop of Algorithm 1.

Another implementation is the MERT module of Phramer[11], an open source MT system written by Marian Olteanu as an alternative to Pharaoh (Koehn, Och, and Marcu, 2003). The MERT module is written in Java, but a quick examination of the package's source code reveals that the `mert` folder contains a whopping 31 Java files! Granted, some of these are class definitions necessary for aspects like evaluation metrics, but the MERT "core" is still a large group of 15–20 files. Compare this to Z-MERT, which consists of only 2 Java files, one of which is a 20-line driver program. This makes compiling Z-MERT almost trivial and running it quite easy.

The biggest drawback with Olteanu's implementation, however, is that it is specifically geared towards Phramer and Pharaoh. It is not immediately clear how one would adapt it for use with other decoders.[12] Z-MERT, on the other hand, can be used immediately as a standalone application, without any modification to the code.

To summarize, Z-MERT is the first MERT implementation specifically meant for public release and for easy use with any decoder. Besides some unique features (Subsection 3.2), there are various advantages to using it: it is extremely easy to compile and run, it produces useful verbose output, it has the ability to resume stopped runs, it is highly optimized (Section 4), and its code is documented `javadoc`-style.

3.2. Z-MERT Usage and Features

Z-MERT is very easy to run. It expects a single parameter, a configuration file:

```
java ZMERT MERT_config.txt
```

The configuration file allows the user to specify any subset of MERT's 20 or so parameters, eight of which are shown in the sample file in Figure 2 (most parameters have default values and need not be specified). This high degree of configurability is Z-MERT's first feature. `-cmd` specifies a one-line file that contains the command that Z-MERT should use to produce an iteration's N-best list (line 5 in Algorithm 1). It is assumed that this command makes use of a decoder config file - `dcfg`, which Z-MERT updates just before producing the N-best list. Z-MERT knows how to update the file because it is informed of the parameter names in the file specified by `-p`.

The `-decOut` parameter indicates the file containing the newly created candidate translations. Z-MERT then proceeds by calculating the sufficient statistics for each candidate, as calculated against the reference translations in the `-r` file. Notice that Z-MERT is agnostic regarding the decoder, and treats it as a black box: Z-MERT prepares the configuration file, starts the decoder, and expects an output file once the decoder is done. The output file, which contains the candidate sentences and feature values, is expected to be in the familiar Moses-

[11] Software and FAQ for Phramer available at: http://www.utdallas.edu/~mgo031000/phramer.

[12] We are assuming here that it is indeed possible to adapt Phramer's MERT module to decoders other than Phramer and Pharaoh. This appears to be the case according to Lane Schwartz, who used it to tune parameters for a third decoder (personal communication). He mentions two Java classes that he needed to write to adapt Phramer's MERT module. We also imagine that, at a minimum, one would have to find and remove `import` and `package` statements referring to the Phramer package.

MERT_config.txt:

```
-cmd    dec_cmd.txt # decoder command file
-dcfg   dec_cfg.txt # decoder config file
-p      params.txt  # parameter file
-decOut nbest.out   # decoder output file
-N      300         # size of N-best list
-r      refs.txt    # reference sentences
-ipi    20          # numInter (see Alg. 1)
-m      BLEU        # evaluation metric
```

params.txt:

```
lm                 |||  1.0  Fix  +0.5  +1.5
phrasemodel pt 0   |||  0.5  Opt  -1    +1
phrasemodel pt 1   |||  0.5  Opt  -1    +1
phrasemodel pt 2   |||  0.5  Opt  -1    +1
wordpenalty        ||| -2.5  Opt  -5    0
```

Parameter name; Initial value; Optimizable?; Range for random values

Figure 2. Sample MERT configuration file and parameter file.

and Joshua-like format.

The -m parameter illustrates another feature of Z-MERT: it is completely modular when it comes to the evaluation metric. Z-MERT can handle any evaluation metric, as long as its sufficient statistics are decomposable. This includes the most popular automatic evaluation metrics in the MT community, such as BLEU and TER . The public release already includes an implementation of BLEU (both IBM and NIST definitions), and implementing a new evaluation metric is quite easy (see Section 5).

4. Experiments

In Figure 3, we report the runtime of a number of experiments to demonstrate Z-MERT's efficiency by optimizing parameters for the Joshua decoder.[13] The MERT optimization was performed on a 2.0 GHz ThinkPad laptop. The development dataset is the text data of MT06, with 4 references per sentence. The evaluation metric being optimized is 4-BLEU (IBM definition). The left graph illustrates the effect of the development set size, for two different N-best sizes. The right graph illustrates the effect of `numInter` of Algorithm 1, for two different set sizes. The reported times are averaged over the first 4 iterations, and are for MERT only and do not include decoding times.

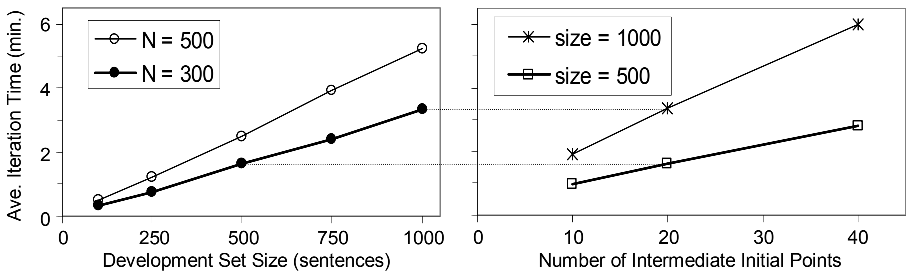

Figure 3. Average iteration time for MERT under different settings.

[13]We use the same parameter file as in Figure 2, except all parameters are optimizable. The language model is a 5-gram LM trained on the English side of the Gigaword corpus (about 130M words). The translation model has about 7.8 million rules.

5. Implementation Details

In addition to the MERT driver and the MERT core, Z-MERT has an `abstract EvaluationMetric` class. For an evaluation metric of interest, we need a corresponding class that `extends EvaluationMetric`. We need only two method definitions: one to calculate sufficient statistics for a candidate, and one that calculates the error given sufficient statistics. Consider the following metric. A candidate is compared against the reference translations. If the first word in the candidate matches the first in any reference, it gets +1, and similarly for the last word. So, a candidate translation can have a score between 0 and 2. Define the error to be the ratio of the sum of those scores divided by the maximum possible score. Here is the implementation:

```
public int[] suffStats(String cand_str, int i) {
  // Calculate the sufficient statistics for cand_str, compared
  // against the references for the ith source sentence.

  int[] retA = new int[suffStatsCount]; // array of SS to be returned
  int firstWordMatches = 0, lastWordMatches = 0;

  for (int r = 0; r < refsPerSen; ++r) {
    if (firstWord(cand_str).equals(firstWord(refSentences[i][r])))
      firstWordMatches = 1;
    if (lastWord(cand_str).equals(lastWord(refSentences[i][r])))
      lastWordMatches = 1;
  }

  retA[0] = firstWordMatches + lastWordMatches;
  retA[1] = 2;

  return retA;
}
public double score(int[] stats) {
  return stats[0]/(double)stats[1];
}
```

For clarity, we omit the trivial definitions for `firstWord(.)` and `lastWord(.)`. Data members such as `refSentences` and `refsPerSen` are already set by the parent class `EvaluationMetric`, which also defines appropriate methods to sum the sufficient statistics. And so, the complexity of the new code is a function of the complexity of the metric itself only; the user need not worry about any kind of bookkeeping, etc.

6. Conclusion and Acknowledgments

We presented Z-MERT, a flexible, fully configurable, easy to use, efficient MERT module for tuning the parameters of MT systems. Z-MERT is agnostic when it comes to the particular system, the parameters optimized, and the evaluation metric. Compiling and running Z-MERT is very easy, and no modification to the source code is needed. The user can easily perform MERT with a new evaluation metric by overriding only a small part of a generic `EvaluationMetric` class.

The author would like to thank the members of the Joshua development team at JHU. This research was supported in part by the Defense Advanced Research Projects Agency's GALE program under Contract No. HR0011-06-2-0001.

Bibliography

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

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL, Demo and Poster Sessions*, pages 177–180.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT-NAACL*, pages 127–133.

Li, Zhifei and Sanjeev Khudanpur. 2008. A scalable decoder for parsing-based machine translation with equivalent language model state maintenance. In *Proceedings of ACL, Second Workshop on Syntax and Structure in Statistical Translation*, pages 10–18.

Och, Franz Josef. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167.

Och, Franz Josef and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of ACL*, pages 295–302.

Venugopal, Ashish and Stephan Vogel. 2005. Considerations in maximum mutual information and minimum classification error training for statistical machine translation. In *Proceedings of the European Association for Machine Translation (EAMT)*.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 89–98

Unsupervised Generation of Parallel Treebanks through Sub-Tree Alignment

Ventsislav Zhechev

Abstract

The need for syntactically annotated data for use in natural language processing has increased dramatically in recent years. This is true especially for parallel treebanks, of which very few exist. The ones that exist are mainly hand-crafted and too small for reliable use in data-oriented applications. In this paper we introduce an open-source system for fast and robust automatic generation of parallel treebanks. We expect the opening of the presented platform to the scientific community to help boost research in the field of data-oriented machine translation and lead to advancements in other fields where parallel treebanks can be employed.

1. Motivation

In recent years much effort has been made to make use of syntactic information in statistical machine translation (MT) systems (Hearne and Way, 2006, Nesson et al., 2006, Lavie, 2008). This has led to increased interest in the development of parallel treebanks as the source for such syntactic data. They consist of a parallel corpus, both sides of which have been parsed and aligned at the sub-tree level.

So far parallel treebanks have been created manually or semi-automatically. This has proven to be a laborious and time-consuming task that is prone to errors and inconsistencies (Samuelsson and Volk, 2007). Because of this, only a few parallel treebanks exist and none are of sufficient size for productive use in any statistical MT application.

In this paper we present an open-source platform for the automatic generation of parallel treebanks from parallel corpora. We discuss algorithms both for cases in which monolingual phrase-structure parsers exist for both languages and for cases in which such parsers are not available. The parallel treebanks created with the methods described in this paper can be used by different statistical MT applications and for translation studies.

We will first discuss the technologies and algorithms used in our system in section 2 and then we will look at the practical details of how to compile and run the system in section 3.

Please cite this article as: Ventsislav Zhechev, Unsupervised Generation of Parallel Treebanks through Sub-Tree Alignment. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 89-98.

2. Algorithms

In this section we introduce a method for the automatic generation of parallel treebanks from parallel corpora. The only tool that is required besides the software presented in this paper is a word-alignment tool (eg. GIZA++ – Och and Ney, 2003). However, if parsers or at least POS taggers exist for any of the languages in question, they can be used to pre-process the data.

In all cases, a word alignment tool is used to first obtain word-alignment probabilities for the parallel corpus in question for both language directions. We will start with the description of the case in which parsers are available for both languages, as this is the core of the system. They are used to parse both sides of the parallel corpus. The resulting parsed data together with the word-alignment probability tables are then used as the input to a sub-tree alignment system that introduces links between nodes in corresponding trees according to their translational equivalence scores. The output of the sub-tree aligner is the desired parallel treebank.

If there is no parser available for one of the languages, the parallel corpus — together with the word-alignment tables — is fed directly to a modified version of the sub-tree aligner that can produce unambiguous parallel treebanks from plain data.

We will now look at the alignment algorithms in greater detail, starting with the tree-to-tree alignment and then moving on to the string-to-string, string-to-tree and tree-to-string cases. A thorough evaluation of the aligner is presented in (Zhechev and Way, 2008).

2.1. Tree-to-Tree Alignment

First, the tree-to-tree aligner has to follow certain principles to fit in the above framework:

- Independence with respect to language pair, constituent-labelling scheme and POS tag set.
- Preservation of the original tree structures.
- Dependence on a minimal number of external resources, so that the aligner can be used even for languages with few available resources.
- The word-level alignments should be guided by links between higher constituents in the trees

These principles guarantee the usability of the algorithm for any language pair in many different contexts. Additionally, there are a few well-formedness criteria that have to be followed to enforce feasible alignments:

- A node in a tree may only be linked once.
- Descendants of a source linked node may only be linked to descendants of its target linked counterpart.
- Ancestors of a source linked node may only be linked to ancestors of its target linked counterpart.

Links produced according to these criteria encode enough information to allow the inference of complex translational patterns from a parallel treebank, including some idiosyncratic translational divergences, as discussed in (Hearne et al., 2007). In what follows, a hypothesised alignment is regarded as incompatible with the existing alignments if it violates any of these criteria.

The sub-tree aligner operates on a per sentence-pair basis in two stages. First, for each possible hypothetical link between two nodes, a translational equivalence score is calculated. Only the links with a nonzero score are stored for further processing. Unary productions from the original trees, if available, are collapsed to single nodes, preserving all labels. Thus the aligner will consider a single node — instead of several nodes — for the same lexical span.

During the second stage, the optimal combination of links is selected from among the available nonzero links using either a greedy-search based, or a full-search based approach.

2.1.1. Translational Equivalence

Given a tree pair $\langle S, T\rangle$ and a hypothesis $\langle s, t\rangle$, we first compute the strings in (1), where $\langle si \ldots s_{ix}\rangle$ and $\langle t_j \ldots t_{jy}\rangle$ denote the terminal sequences dominated by s and t respectively, and $\langle S_1 \ldots S_m\rangle$ and $\langle T_1 \ldots T_n\rangle$ denote the terminal sequences dominated by S and T. Here, *inside* are the strings that represent the spans of the nodes being linked and *outside* are the strings that lay outside the spans of those nodes.

(1)
$$\begin{array}{ll} \texttt{inside} & \texttt{outside} \\ s_l = \langle s_i \ldots s_{ix}\rangle & \overline{s_l} = \langle S_1 \ldots s_{i-1}\, s_{ix+1} \ldots S_m\rangle \\ t_l = \langle t_j \ldots t_{jy}\rangle & \overline{t_l} = \langle T_1 \ldots t_{j-1}\, t_{jy+1} \ldots T_n\rangle \end{array}$$

(2)
$$\gamma(\langle s, t\rangle) = \alpha(s_l|t_l) \cdot \alpha(t_l|s_l) \cdot \alpha\left(\overline{s_l}|\overline{t_l}\right) \cdot \alpha\left(\overline{t_l}|\overline{s_l}\right)$$

(3) score1
$$\alpha(x|y) = \prod_{j}^{|y|} \sum_{i}^{|x|} P(x_i|y_j)$$

(4) score2
$$\alpha(x|y) = \prod_{i}^{|x|} \frac{\sum_{j}^{|y|} P(x_i|y_j)}{|y|}$$

The score for the given hypothesis $\langle s, t\rangle$ is computed using (2) and (3) or (4). According to (3), for each source token we first sum the word-alignment probabilities of the target tokens, given the source token. This gives us the probability masses of the target string corresponding to each of the source tokens and multiplying these gives us the alignment probability. In (4), the word-alignment probabilities are used to get an average vote by the source tokens for each target token. Then the product of the votes for the target words gives the alignment probability for the two strings. The final translational equivalence score is the product of the alignment probabilities for the inside and outside strings in both language directions as in (2).

2.1.2. Greedy-Search Algorithm

The greedy-search algorithm is very simple. The set of nonzero-scoring links is processed iteratively by linking the highest-scoring hypothesis at each iteration and discarding all hypotheses that are incompatible with it until the set is empty.

Problems arise when there happen to be several hypotheses that share the same highest score. There are two distinct cases that here: these top-scoring hypotheses may or may not represent incompatible links. If all such hypotheses are compatible, they are all linked at the same time; otherwise these hypotheses are skipped and processed at a later stage.

The sub-tree aligner can be built to use one of two possible skipping strategies, which we will call *skip1* and *skip2*. According to the *skip1* strategy, hypotheses are simply skipped until a score is reached, for which only one hypothesis exists. This hypothesis is then linked and the selection algorithm continues as usual. The *skip2* strategy is more complex, in that we also keep track of which nodes take part in the skipped hypotheses. Then, when a candidate for linking is found, it is only linked if it does not include any of these nodes.

Regardless of whether *skip1* or *skip2* is used, sometimes a situation occurs in which the only hypotheses remaining unprocessed are equally likely candidates for linking. In such ambiguous cases our decision is not to link anything, rather than make wrong a decision.

During initial testing of the aligner we found that often lexical links would get higher scores than the non-lexical links,[1] which sometimes resulted in poor lexical links blocking bona fide non-lexical ones. To address this issue, an extension to the selection algorithm was developed, which we call *span1*. When enabled, this extension results in the set of nonzero hypotheses being split in two subsets: one containing all hypotheses for lexical links, and one containing the hypotheses for non-lexical links. Links are then first selected from the second subset, and only when it is exhausted does the selection continue with the lexical ones.

2.1.3. Full-Search Algorithm

This is a backtracking recursive algorithm that enumerates all possible combinations of non-crossing links. All maximal combinations[2] found during the search are stored for further processing. After the search is complete, the probability mass of each maximal combination is calculated by summing the translational equivalence scores for all the links in the it and the one that has the highest probability mass is selected as the best alignment for the sentence pair.

Often, there are several distinct maximal combinations that share the highest probability mass. The disambiguation strategy that we currently employ is to take the largest common subset of all maximal combinations.

2.2. Other Alignment Modules

In this section we look at the string-to-string, tree-to-string and string-to-tree modules that are used when a parser is not available for one or both of the languages being aligned.

The string-to-string aligner can accept as its input plain or POS-tagged data. For a pair of sentences, all possible binary trees are first constructed for each sentence. All nodes in these trees have the same label (X) and are used as available link targets. In the case of POS-tagged data, the pre-terminal nodes receive the POS tags as labels.

After all link-hypothesis scores have been calculated, the string-to-string aligner continues with the selection of links in the same manner as the sub-tree aligner, with one extension; after a link has been selected — besides all incompatible links — all binary trees that do not include

[1] *lexical* are such links, for which at least one of the linked nodes spans over only one word.

[2] A *maximal combination* of non-crossing links is a combination of links for which any newly added link would be incompatible with at least one of the links already in the combination.

the linked nodes are discarded with any nonzero hypotheses attached to them. In this way, only those binary trees that are compatible with the selected links remain after the linking process.

In an additional step for the string-to-string aligner, all non-linked nodes (except for the root nodes) are discarded, thus allowing for the construction of unambiguous *n*-ary trees for the source and target sentences. If necessary, non-linked nodes are left intact to provide supporting structure in the trees. It is also possible to output a parse forest of all binary trees that are compatible with the alignments.

In its operation, the string-to-string aligner is very similar to ITG (Wu, 2000), however its goal is the generation of a parallel treebank, rather than the induction of a bilingual grammar.

The tree-to-string and string-to-tree modules differ from the string-to-string module in that a parser is available for one of the languages being aligned. In this case, the available parses are used, where available, rather than generate hypothetical binary trees. Also, at the output stage, the existing parses are preserved, except for any unary productions that are being collapsed as in the tree-to-tree alignment module. The non-parsed side may be POS-tagged, if a POS tagger is available.

2.3. Re-scoring

It can be argued that each newly induced link in a sentence pair should affect the decisions regarding which links to select further in the alignment process for this sentence pair. This can be simulated to a certain extent using the simple re-scoring module discussed in this section.

The operation of this module relies on the fact that after a link has been introduced for a pair of trees, some of the word alignments available in the word-alignment tables for the tree pair will be incompatible with this link. Namely, these are alignments between words within the span of the source node being linked and words without the span of the target node; as well as alignments between words without the source node and words within the target node.

Thus, each time a new link has been selected, the incompatible word alignments are removed from the list of available word alignments for the tree pair and the scores of the remaining link hypotheses are recalculated. The linking process then continues as usual.

3. Usage

The distribution package of the aligner consists of a single `bzip2` compressed tarball. The system is implemented using standard C++ and can be compiled using GCC version 4.0 and higher. The source code is distributed with a `configure` script, which handles the configuration options. After the distribution package is unpacked, this script can be found in the `build` sub-folder. Run `./build/configure --help` for a full list of compilation options. A `README` file is included with the distribution, which includes an up-to-date version of the information presented in this paper.

I suggest configuration and compilation in the `build` folder. Configuration and compilation in other folders has not been tested and is discouraged. To speed up reconfiguration, I suggest passing the argument `-C` to the `configure` script, which will turn on caching.

Run `./configure [options]` to configure the tools you want to compile. There is an option for the `configure` script that controls which tools are to be compiled and installed: `--enable-tools="<list of tools>"` By default, only the tree-to-tree aligner is compiled and installed. To install all available tools, use `--enable-tools=all`. If you want to specify precisely which tools are to be built and installed, use `align` for the standard tree-to-tree aligner; `lattice` for a full-search based tree-to-tree aligner (experimental); `str2str` for a string-to-string, tree-to-string and string-to-tree 3-in-1 alignment module.

Run `make && make install` to compile and install the software. The default installation destination is `/usr/local`, but it can be changed using `configure` options.

Also, if you have GCC 4.2 or later, you can compile the aligner for parallel execution. To configure the software for parallel execution, supply the `--enable-parallel` option to `configure`. When compiled for parallel execution, the `OMP_DYNAMIC` environment variable controls the behaviour of the software. If you set this variable to `FALSE`, the software will use all available CPUs on your system, regardless of whether there are other processes running or not. If you are running other resource intensive tasks on your system you may want to set `OMP_DYNAMIC` to `TRUE`. In this case, the software will decide dynamically what amount of resources to use without interfering with other running processes.

3.1. Tree-to-Tree Aligner

The options controlling the functionality of the aligner have defaults that can be changed by passing options to the `configure` script. Here is a list of the different options and their function:

`--enable-data-set=<data_set_name>` This option should be set to a string, describing the data set it will operate on. By default this option is set to `"unknown"`.

`--disable-span1` If you supply this option to the configure script, the aligner will be compiled without the *span1* feature. By default, this feature is turned on.

`--enable-score={1, 2}` You can choose the scoring mechanism that is to be used by the aligner by using this option. By default, the aligner will use *score2*.

`--enable-skip={1, 2}` You can choose the selection algorithm that is to be used by the aligner by using this option. By default, the aligner will use *skip2*.

`--enable-rescoring` This option turns on the re-scoring module. It is off by default.

`--enable-lowercasing` This option should be defined, if you are using lowercased word-alignment data. It is disabled by default.

`--enable-log-based-probabilities` If you turn on this option, the link hypothesis scores will be stored as logarithms. The option is on by default.

You would normally run the aligner in one of the following two ways:

```
align <source_to_target_lex_probs> <target_to_source_lex_probs> [<source_to_target_phrase_probs>] <input_corpus>
align <config_file>
```

You should always supply the proper command line arguments, as they are not checked for correctness. Here is a description:

`<source_to_target_lex_probs>` The path to the file which holds the source-to-target word alignment probabilities. The format is `<target> <source> <probability>\n`

<target_to_source_lex_probs> The path to the file which holds the target-to-source word alignment probabilities. The format is <source> <target> <probability>\n

<source_to_target_phrase_probs> The path to the file which holds the source-to-target phrase alignment probabilities. This file is currently used only to calculate some statistics and you can safely omit it, as its use slows down the system and increases the memory footprint.

<input_corpus> The path to the file containing the aligned parsed sentences or –. Supplying – for this parameter will direct the aligner to read data from the standard input, rather than from a file. The format is <source>\n<target>\n\n\n. The parsed sentences should be in bracketed format, using (and) as delimiters. White-space (except new lines) is irrelevant and any character is allowed in both terminal and non-terminal nodes (except spaces; spaces are not allowed in non-terminal nodes and signify multiword units in terminal nodes).

<config_file> The path to a file containing run-time options, one option per line. This file has the format <option_name> <option_value>\n. Any line starting with a # character will be ignored. You can specify the following options in the file that correspond to command line options: input — corresponds to <input_corpus>; source_alignments — <source_to_target_lex_probs>; target_alignments — <target_to_source_lex_probs>; phrase_alignments — <source_to_target_phrase_probs>. Additionally, the input option may be omitted in which case the aligner will read data from the standard input. There are some additional options that may be specified in the configuration file, but are not required. output is used to specify the path to a file in which the output of the aligner is to be written. Information about the output format is given later in this section. log is used to specify the path to a file in which run-time information and statistics are to be written. expensive_statistics can be set to all, none, POS or search and controls whether certain memory-expensive statistics should be calculated. When not specified, this option defaults to all. The statistics in question concern the distribution of POS tags and POS tag-pairs and keeping track of the search-space reductions during alignment.

If you use command line options when running the aligner, or use a configuration file but do not specify the output and log options, all output is sent to the standard output. If you specify only the output option in the configuration file, the output of the aligner will be written to the file specified, while the performance statistics will be written to the standard output. If you, on the other hand, specify only the log option, the statistics will be written to the specified file and the output will go to standard output. In case you specify both options, both the output and the statistics will be written to the corresponding files. The format of the output for the parallel treebank is <source>\n<target>\n<source_node_id> <target_node_id> … \n\n. The non-terminal nodes in the parsed trees all have IDs attached with a – character. These IDs are used to represent the links between the nodes of the trees. An alignment example is shown in Figure 1 together with the proper input and output.

If you compile the lattice tool, it will use the exact same options as the tree-to-tree aligner and will produce output in the same format. The most significant difference is that it will use the full-search algorithm for the induction of the sub-tree alignments, rather than the greedy-search based algorithm. This tool is still experimental, though, and due to the combinatorial nature of the full-search algorithm may not find a solution for all sentence pairs within an ac-

ceptable timeframe. Because of this the use of this tool is strongly discouraged. The `lattice` tool does not support the *span1* extension yet and the *skip** modules are irrelevant to it.

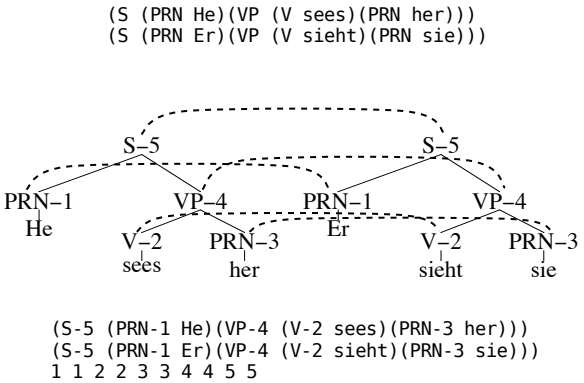

Figure 1: An aligned tree pair with the corresponding system input and output

3.2. String-to-String Aligner

Here only the differences between the string-to-string aligner and the tree-to-tree aligner will be listed. Anything not mentioned works exactly as described for the tree-to-tree aligner, i.e. the compilation and configuration options available for the tree-to-tree aligner are also available here.

The aligner should be run with command line arguments. You would normally run it in one of the following two ways:

```
   align_str2str <operation_mode> <input_type> <output_type> <source_to_target_lex_probs>
<target_to_source_lex_probs> [<source_to_target_phrase_probs>] <input_corpus>
   align_str2str <config_file>
```

Here is the description of the options:

`<operation_mode>` This argument specifies the mode of operation of the aligner and cannot be omitted. `str2str` will evoke standard string-to-string alignment. In case a parser is available for one of the languages being aligned, the aligner can be set to run in string-to-tree or tree-to-string mode. The parameters for these modes are `str2tree` and `tree2str` respectively. In these cases you have to make sure that the correct side of the corpus contains bracketed representations of parsed sentences. The format of the other side of the corpus is controlled by the `<input_type>` argument.

`<input_type>` If you supply POS-tagged sentences, this argument should be `tagged` and for plain sentences this should be `plain`. This argument cannot be omitted.

`<output_type>` This argument is used to select the type of output of the aligner and cannot be omitted. There are three possible options: `standard`, `parse` and `XML`. The `standard` output has the format presented in Figure 2 for each sentence pair. If the `mother_node_ID` of a node is `0`, then this node has no ancestors (it is a root node). There may be more than one

such node for each sentence. This format preserves enough nodes to represent all possible binary trees for the sentences in the pair that are consistent with the induced links. The `parse` and `XML` output formats present minimal trees, consisting only of the pre-terminal nodes and the linked nodes for the sentences in each pair. In case there is more than one root node for a particular tree, an extra node with label *X* and ID `100000` is inserted as the mother of all root nodes. Both formats give a standard bracketed representation of unambiguous parse trees.

```
#BOP
#BOS
<word1>\t<mother_node_ID>
<word2>\t<mother_node_ID>
…
#<node_ID> <node_label>\t<mother1_node_ID> <mother2_node_ID> …
…
#EOS
#BOS
…
#EOS
#LINKS <source1_node_ID> <target1_node_ID> …
#EOP

#BOP
…
#EOP
```

Figure 2: Standard output format of the string-to-string aligner

`<input_corpus>` There are two possible formats for the sentences, while the overall file format remains as for the tree-to-tree aligner. The first format is simply `<word1> <word2> … <wordn>`. The second format is `((<word1>)) ((<word2>)) … ((<wordn>))` and can be used to specify the boundaries of multiword units. This second format can also be used for supplying POS tags for the words of the sentences. In that case the format is `((<word1> <POS1>)) ((<word2> <POS2>)) … ((<wordn> <POSn>))`. A specific requirement for the use of the string-to-string aligner is the existence of one of two open source modules on your system: If you are using the first input format, you need the Boost Tokenizer library; If you are using the second input format, you need the Boost Regex library.

`<config_file>` The path to a file containing run-time options, one option per line. The same rules apply as for the config file for the tree-to-tree aligner. There are three additional options, however: `operation_mode`, `input_type` and `output_type`. They correspond directly to their command-line counterparts.

4. Conclusion

We have presented a novel platform for the fast and robust automatic generation of parallel treebanks. The algorithms described are completely language pair-independent and require a minimal number of resources; besides a parallel corpus, a word alignment tool is the only extra software required. If available, POS taggers or monolingual phrase-structure parsers can be used to pre-process the data.

The software is distributed as C++ source code together with a script for configuring the compilation process and extensive documentation. The latest version can be downloaded from http://ventsislavzhechev.eu/Home/Software/Software.html.

Acknowledgments

We would like to thank Mary Hearne, John Tinsley, Andy Way and Khalil Sima'an for their participation in the development of the algorithms. The current work is part of the ATTEMPT project at NCLT, DCU, Ireland and is generously supported by Science Foundation Ireland (grant number 05/RF/CMS064).

Bibliography

Hearne, Mary and Andy Way. 2006. Disambiguation Strategies for Data-Oriented Translation. In *Proceedings of the 11th Conference of the European Association for Machine Translation (EAMT '06)*, pp. 59–68. Oslo, Norway.

Hearne, Mary, John Tinsley, Ventsislav Zhechev and Andy Way. 2007. Capturing Translational Divergences with a Statistical Tree-to-Tree Aligner. In *Proceedings of the 11th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI '07)*, eds. Andy Way and Barbara Gawronska, pp. 85–94. Skövde, Sweden: Skövde University Studies in Informatics.

Lavie, Alon. 2008. Stat-XFER: A General Search-based Syntax-driven Framework for Machine Translation. In *Proceedings of the 9th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing '08)*, ed. Alexander F. Gelbukh, pp. 362–375. Vol. 4919/2008 of *Lecture Notes in Computer Science*. Haifa, Israel: Springer.

Nesson, Rebecca, Stuart M. Shieber and Alexander Rush. 2006. Induction of Probabilistic Synchronous Tree-Insertion Grammars for Machine Translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA '06)*, pp. 128–137. Boston, MA.

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

Samuelsson, Yvonne and Martin Volk. 2007. Alignment Tools for Parallel Treebanks. In *Data Structures for Linguistic Resources and Applications: Proceedings of the Biennial GLDV Conference 2007*, eds. Georg Rehm, Andreas Witt and Lothar Lemnitzer. Tübingen, Germany: Gunter Narr.

Wu, Dekai. 2000. Bracketing and aligning words and constituents in parallel text using Stochastic Inversion Transduction Grammars. In *Parallel Text Processing: Alignment and Use of Translation Corpora*, ed. Jean Veronis, chap. 7. Dordrecht: Kluwer.

Zhechev, Ventsislav and Andy Way. 2008. Automatic Generation of Parallel Treebanks. In *Proceedings of the 22nd International Conference on Computational Linguistics (CoLing '08)*, pp. 1105–1112. Manchester, UK.

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009

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 intersting 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`.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009

LIST OF AUTHORS

Peter Berck
Tilburg centre for Creative Computing
University of Tilburg
P. O. Box 90153
5000 LE Tilburg, The Netherlands
p.j.berck@uvt.nl

Nicola Bertoldi
FBK - Fondazione Bruno Kessler
via Sommarive 18
38100 Povo, Trento, Italy
bertoldi@fbk.eu

Antal van den Bosch
Tilburg centre for Creative Computing
University of Tilburg
P. O. Box 90153
5000 LE Tilburg, The Netherlands
antal.vdnbosch@uvt.nl

Chris Callison-Burch
The Center for Language
and Speech Processing
Computational Science
and Engineering Building
3400 North Charles Street
Baltimore, MD 21218, USA
ccb@cs.jhu.edu

Kevin Donnelly
Llanfairpwll
Ynys Môn. LL61 6UX
Wales
kevin@dotmon.com

Jean-Baptiste Fouet
Systran SA
La Grande Arche 1
Parvis de la Défense
92044 Paris, La Défense cedex
France
fouet@systran.fr

Kuzman Ganchev
Computer & Information Science
University of Pennsylvania
3330 Walnut Street
Philadelphia, PA 19104-6389, USA
kuzman@cis.upenn.edu

Josef van Genabith
Center for Next Generation Localisation
Dublin City University
Dublin 9, Ireland
josef@computing.dcu.ie

João Graça
L2F - Inesc-ID Lisboa
Av. Rovisco Pais 1049-001
Lisboa, Portugal
joao.graca@l2f.inesc-id.pt

Yvette Graham
National Centre for Language Technology
Dublin City University
Dublin 9, Ireland
ygraham@computing.dcu.ie

Barry Haddow
School of Informatics
University of Edinburgh
Informatics Forum
10 Crichton Street
Edinburgh, EH8 9AB, Scotland
bhaddow@inf.ed.ac.uk

Sanjeev Khudanpur
The Center for Language
and Speech Processing
Computational Science
and Engineering Building
3400 North Charles Street
Baltimore, MD 21218, USA
khudanpur@jhu.edu

Zhifei Li
The Center for Language
and Speech Processing
Computational Science
and Engineering Building
3400 North Charles Street
Baltimore, MD 21218, USA
zhifei.work@gmail.com

Ben Taskar
Computer & Information Science
University of Pennsylvania
3330 Walnut Street
Philadelphia, PA 19104-6389, USA
taskar@cis.upenn.edu

Wren Thornton
The Center for Language
and Speech Processing
Computational Science
and Engineering Building
3400 North Charles Street
Baltimore, MD 21218, USA
wren@freegeek.org

Francis M. Tyers
Grup Transducens
Departament de Llenguatges
i Sistemes Informàtics
Universitat d'Alacant
E-03080 Alacant, Spain
ftyers@prompsit.com

Ashish Venugopal
Google Inc.
ashishv@cs.cmu.edu

Omar F. Zaidan
Johns Hopkins University
3400 N. Charles Street
Computer Science Department - 224 NEB
Baltimore, MD 21218, USA
ozaidan@cs.jhu.edu

Ventsislav Zhechev
NCLT, School of Computing
Dublin City University
Glasnevin
Dublin 9, Ireland
contact@ventsislavzhechev.eu

Andreas Zollmann
Carnegie Mellon University
407 South Craig Street
Pittsburgh, PA 15213, USA
zollmann@cs.cmu.edu