PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 106 OCTOBER 2016 159–168

Otedama: Fast Rule-Based Pre-Ordering for Machine Translation

Julian Hitschler[a], Laura Jehl[a], Sariya Karimova[ac], Mayumi Ohta[a], Benjamin Körner[a], Stefan Riezler[ab]

[a] Computational Linguistics, Heidelberg University, Germany
[b] IWR, Heidelberg University, Germany
[c] Kazan Federal University, Russia

Abstract

We present Otedama,[1] a fast, open-source tool for rule-based syntactic pre-ordering, a well established technique in statistical machine translation. Otedama implements both a learner for pre-ordering rules, as well as a component for applying these rules to parsed sentences. Our system is compatible with several external parsers and capable of accommodating many source and all target languages in any machine translation paradigm which uses parallel training data. We demonstrate improvements on a patent translation task over a state-of-the-art English-Japanese hierarchical phrase-based machine translation system. We compare Otedama with an existing syntax-based pre-ordering system, showing comparable translation performance at a runtime speedup of a factor of 4.5–10.

1. Introduction

Syntactic pre-ordering is a commonly used pre-processing technique in state-of-the-art machine translation systems. It attempts to adjust the syntax of the source language to that of the target language by changing the word order of a source sentence prior to translation. Originally, this technology was devised to overcome a weakness of classical phrase-based translation systems (Koehn et al., 2003), which usually penalize moving target phrases far away from their source positions. This is a major

[1] An open-source version is available at `https://github.com/StatNLP/otedama`. Otedama is named after a Japanese juggling game.

 Corresponding author: `riezler@cl.uni-heidelberg.de`
Cite as: Julian Hitschler, Laura Jehl, Sariya Karimova, Mayumi Ohta, Benjamin Körner, Stefan Riezler. Otedama: Fast Rule-Based Pre-Ordering for Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 106, 2016, pp. 159–168. doi: 10.1515/pralin-2016-0015.

source of errors when translating between languages with heterogeneous and dissimilar sentence structures. Hierarchical phrase-based translation systems do not place a similar prior penalty on phrase reordering during decoding, however, such systems have been shown to profit from syntactic pre-ordering as well (de Gispert et al., 2015).

Otedama implements a variant of the approach of Genzel (2010), which learns cascading lists of rules for syntactic transformation. Unlike early works on pre-ordering which rely on hand-written rules (Collins et al., 2005), Otedama automatically extracts rules from parse trees. While recent work has applied other learning algorithms to the problem of learning pre-ordering models (Lerner and Petrov, 2013; Jehl et al., 2014; de Gispert et al., 2015; Nakagawa, 2015), automatic rule-learning is a popular and suitable choice for pre-ordering systems because syntactic features are discrete and relatively dense and the resulting models allow for very fast application to system input. In particular, the rule-based approach deals well with the combinatorial explosion that is incurred when attempting to train in-domain pre-ordering models on data with a high prevalence of long sentences, as we demonstrate in our system evaluation. Furthermore, our approach is compatible with nearly any external parsing framework, in difference to approaches where preordering and parsing need to be tightly connected for improved performance (Nakagawa, 2015). Despite the fact that pre-ordering continues to be an important technique in high-quality machine translation, so far, there is a lack of an open-source implementation of learners and online application systems for pre-ordering that are convenient to use and fast enough to be suitable for rapid prototyping and meaningful comparison of new approaches to existing baselines. We created Otedama to address this lack. We compare Otedama to two variants of an open-source pre-orderer by Neubig et al. (2012) which induces a bracketing transduction grammar for producing a re-ordered string. Our system yields comparable improvements in translation quality at a runtime speedup of a factor of 4.5–10. Our tool is available as open-source code and compatible with nearly any external parser.

2. Specifications

2.1. Model Formalism

Our model formalism follows the work of Genzel (2010). The model is trained based on syntactic parse trees of the source sentences in a parallel corpus, in addition to a bilingual word alignment. Parse trees are obtained from an automatic dependency parser. By introducing head nodes, non-projective dependency graphs are converted to a tree format (see Figure 2a). In order to obtain good results, the parser should produce labeled dependencies. Otedama provides bindings to the Stanford Parser and is fully compatible with any parser that produces POS tags and dependency labels in the `CoNLL` output format,[2] such as, for example, the Parzu parser for

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

```
function ExampleRule(Node N)
    if N.tag = _VBD AND
      N.dep = root AND
      N.parent.tag = ROOT AND
      N.children[1].tag = VBD AND
      N.children[1].dep = head AND
      N.children[2].tag = _NN AND
      N.children[2].dep = dobj then
        swap(N.children[1], N.children[2])
    end if
end function
```

Figure 1: Example pre-ordering rule. This rule swaps a past tense verb (VBD) with a noun phrase (_NN).

German (Sennrich et al., 2009).[3] Thus, Otedama is able to process a wide variety of source languages.

The rules learned by Otedama comprise a matching context for nodes in a parse tree, expressed in terms of the POS tags and dependency labels of a node, its parent node, and a sequence of neighboring children of the node. Furthermore, a reordering operation is defined on the sequence of neighboring children, which permutes the positions of the children of the node, thereby making syntactic changes to the source language corpus with the goal of approximating the target language syntax. An example rule is given in Figure 1. Rules are learned iteratively, meaning that a rule which is found useful (i.e. that increases the alignment monotonicity of the training corpus) is first applied to the entire corpus before further rules are tested and applied. This results in a cascading list of rules, which can be applied to source language parse trees before translation.

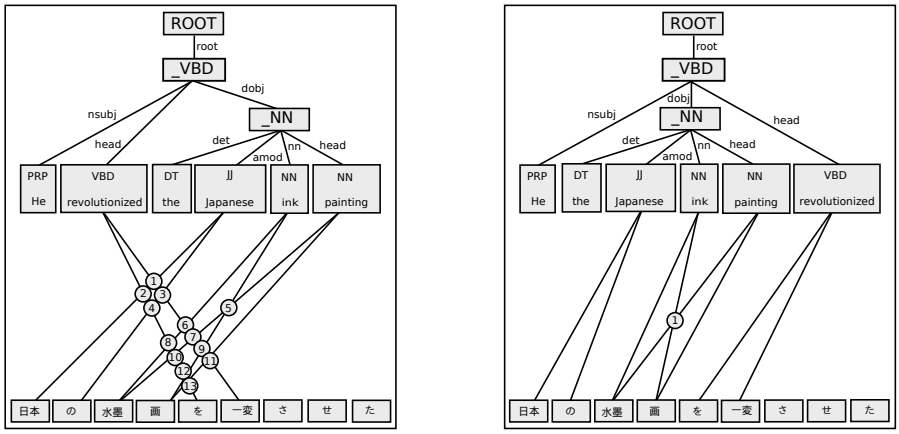
In order to avoid combinatorial explosion, the permutations are restricted to a sliding window, the size of which can be specified to be either 2, 3, or 4 (henceforth referred to as parameter l). For a window size of four, child node reordering is therefore restricted to children that are at most three nodes apart from each other for any one rule.

2.2. Training Procedure

We follow the basic procedure delineated by Genzel (2010) for learning pre-ordering rules, with some modifications. We first describe the basic training procedure.

This objective of the training procedure is to minimize the number of alignment crossings (Genzel, 2010) in a parallel training corpus. For example, the sentence pair shown in Figure 2a has 13 alignment crossings. Applying the example rule from Figure 1 reduces the number of alignment crossings to 1, as shown in Figure 2b. This

[3] https://github.com/rsennrich/ParZu



(a) Before rule application

(b) After rule application

Figure 2: Example parse tree with POS tags, dependency labels and alignment crossings (shown as numbered dots). The left side shows the original tree, the right side shows the same tree after applying the rule from Figure 1. This rule swaps the second and third child node of the "`_VBD`"-node, resulting in a reduction of alignment crossings from 13 to 1. Some alignment links were omitted for clarity.

metric is trivial to compute for any word alignment and can be used as a proxy measure for evaluating pre-ordering models.

Training proceeds iteratively until a convergence criterion is reached or a fixed number of iterations or amount of runtime have elapsed. Each iteration carries out a two-step procedure: In the first step, all possible candidate rules from a small, random subset of the training corpus are generated. This is done by extracting all rule contexts (consisting of POS-tags and dependency labels of the current node, its parent, and a span of its child nodes up to the window size), and pairing them with all possible permutations of the given nodes for all dependency trees in the subset. Only such candidate rules which locally increase alignment monotonicity are generated. In the second step, each candidate rule is evaluated by applying it to the entire training corpus (or, in Genzel's case, a second, larger subset thereof) and measuring the reduction of alignment crossings. The candidate rules which increase the alignment monotonicity of the training corpus and fulfill certain variance constraints are added to the final rule set and applied to all training data. The partially reordered training set is then used as input to the next iteration in which a new random subset is sampled for rule candidate generation. This iterative procedure has two advantages: First, it is likely to extract the most general rules, i.e. the rules which have the largest effect on all

training examples in the beginning, and then subsequently proceed to more specific rules, because sentences with frequent syntactic constructions are more likely to be sampled than those with infrequent ones. This means that if training time is limited, the most important rules will still be extracted. Second, even though the rule is restricted to a window of a small number of nodes, the iterative procedure allows for rules to be applied sequentially to the child nodes of the same parent, thus achieving long-range permutations.

The following paragraphs describe our modifications of Genzel's training recipe.

Scalability Genzel's approach was designed for a MapReduce architecture, since the rule extraction and evaluation at each training iteration can be easily parallelized. Training on a large training set, however, requires a large parallel infrastructure running MapReduce. Our implementation aims to eliminate the dependency on MapReduce, allowing the user to run Otedama on a single machine, and to provide a flexible solution which can be adjusted for the available resources. We achieve these goals by dynamically adjusting the size of the sampled subset of the training instances which is used to generate candidate rules at each iteration. Since the candidate generation step is much more expensive than the rule evaluation, we can still calculate crossing alignment reduction on the entire training set, allowing for good generalization. The initial sample size (henceforth referred to as parameter m) can be set by the user. If the number of rules learned in the last iteration is below a minimum value of 20 or above a maximum value of 1000, the sample size is adjusted in the following iteration by doubling or halfing it. The candidate rule generation and scoring then follows Genzel's recipe. Our implementation supports multi-threading on a single machine.

Variance Constraints Depending on the quality of the parser and the syntactic complexity of the training data, feedback on the effect of rules from reduction in alignment crossings can be quite noisy. Otedama provides the option of specifying a variance constraint for rule candidates in the form of a minimum ratio between sentences in the training data on which alignment crossing is reduced, compared to those on which it increases (henceforth referred to as parameter v). For example, setting $v = 2$ means that only such rules that increase the alignment monotonicity of at least two times the number of sentences than the number of sentences for which alignment monotonicity decreases should be retained during training.

Rule Application Otedama provides two options for making rule application more flexible: At training time, it can be specified whether rules with feature sets that are subsets of the matching context should also be extracted as candidates. For a rule extracted from a node with two children, where the original matching context comprises eight features (the POS-tag and dependency labels of the parent, the node, and the two children, respectively), this means that all 254 nonempty subsets of the origi-

nal matching context are extracted and evaluated as candidate rules. After candidate rules are created from a node, they are ordered increasingly by the size of their feature sets. This means that Otedama first evaluates the most general rules with the fewest required features and, if no rule is found to increase alignment monotonicity and fulfill the specified variance constraints, proceeds to try the more specific rules. In addition, we allow fuzzy matching of rules by specifying the maximum number of features to be matched globally, at both training and test times.

3. Evaluation

As demonstrated by de Gispert et al. (2015), it is potentially beneficial to train pre-ordering models on in-domain data rather than to deploy a model trained on a different domain. For our evaluation we selected the challenging domain of patents. Patents contain very long, convoluted sentences, specialized vocabulary and idioms. They are thus a poorer fit for a general purpose parser than other data. However, we will show that pre-ordering improves translation quality on translation from English into Japanese (EN-JA) and from German into English (DE-EN).

Baseline SMT system All our systems use the hierarchical phrase-based paradigm (Chiang, 2007) as implemented by the `cdec` decoder (Dyer et al., 2010). Our English-Japanese system was trained on the NTCIR7 patent MT data set by Utiyama and Isahara (2007) (1.8M sentence pairs). We used the official development sets dev1 and dev2 for tuning, while reserving dev3 for evaluation. The Japanese side was segmented using MeCab[4]. The English side was tokenized and true-cased using scripts from the Moses toolkit[5]. Our German-English system was trained on 750K sentence pairs from the PatTr corpus (Wäschle and Riezler, 2012). Both sides of the data set were tokenized and true-cased. Both systems used MGIZA++[6] for word alignment. The alignments were produced by the training script provided in the Moses toolkit with the default number of IBM Model 1–4 and HMM iterations (5 iterations of IBM Model 1 and HMM Model, 3 iterations IBM Model 3 and 4). Alignments were symmetrized using the grow-diag-final-and heuristic. We trained a 5-gram target side language model using `lmplz` (Heafield et al., 2013). Rule extraction was performed using `cdec`'s default parameters (maximum rule span = 15, maximum number of symbols per rule = 5). Our system was tuned with the pairwise ranking optimizer `dtrain` (Simianer et al., 2012). Tuning was run for 15 iterations, using a k-best size of 100 and a constant learning rate of 0.00001. Final weights were obtained by averaging over all

[4] `http://taku910.github.io/mecab/`

[5] `https://github.com/moses-smt/mosesdecoder`

[6] `http://www.cs.cmu.edu/~qing/giza/`

system	# crossing alignments	% of baseline
Baseline	1840536	100
Otedama	1465120	79.60
Lader class	1447312	78.64
Lader full	1364459	74.13

Table 1: EN-JP crossing scores

iterations. Both tuning and decoding used a cube pruning pop-limit of 200. We report BLEU scores we calculated using `MultEval` (Dyer et al., 2011) on tokenized output.

Pre-orderer training Pre-ordering models were trained on 100,000 parallel sentence pairs from our parallel training data. [7] English source data was parsed with the Stanford Parser (Socher et al., 2013), German source data with the Parzu parser (Sennrich et al., 2009). In contrast to Genzel (2010), we used IBM Model 4 alignments instead of IBM Model 1 alignments in order to reduce noise on our smaller data set. We re-used the symmetrized word alignments that were created during baseline training, as described in the previous paragraph. We tried out various configurations of Otedama's hyper-parameters window size $l \in \{3, 4\}$ and variance constraints $v \in \{0, 2, 5, 10\}$. For both language pairs, $l = 3$ and $v = 2$, without fuzzy rule matching or feature subsets performed best. The number of matching features was therefore set to the maximum value of 10.[8]The initial subsample size for rule candidate generation was kept constant at $m = 10$ throughout our experiments. Training was stopped after exhaustion of a fixed runtime budget. The rules extracted by Otedama were applied to our MT training and testing data, and word alignments of the training data were re-calculated using GIZA++ with the same settings as above. Hierarchical rule extraction and tuning were then carried out in the standard way, using the same settings as our baseline system.

Instructions for training and applying Otedama, along with a small example script and recommended parameter values, are provided on the GitHub page.

Comparative pre-ordering system We compare our system to `Lader` (Neubig et al., 2012).[9] `Lader` is an open-source reordering tool for machine translation. It performs a

[7]The size of the data set was chosen due to the runtime constraints of our comparison system in order to ensure meaningful comparison. Due to its parallelized architecture, Otedama could process significantly larger data sets during learning with only negligible overhead costs.

[8]This is the maximum number for a window size of 3. With a window of 4, there would be at most 12 features.

[9]`https://github.com/neubig/lader`

large-margin training treating the parser's derivation tree as a latent variable. `Lader` allows to define various features. One possibility is to train a pre-ordering model only from the parallel text and word classes (*Lader class*). Another option is to enhance the model with additional linguistically informed features (POS tags and parse trees), if available (*Lader full*). In both cases, the phrase table was also used as a feature to keep frequent contiguous phrases. In our experiments we have replicated the standard feature set from Neubig et al. (2012) using feature templates. To calculate classes we utilized word classes computed by GIZA++. To obtain POS tags and parse trees the Stanford tagger and the Stanford lexicalized PCFG parser were used, for both English and German as source languages. We trained our models with the default learner Pegasos. Due to the time constraints training was stopped after 10–15 iterations and the model with the best alignment crossing score selected. We did not observe improvements in alignment monotonicity on the training data past 5 iterations. Training times were comparable for Otedama and Lader models, depending on the exact Lader model specifications.

Intrinsic Evaluation by Crossing Score The crossing score counts the number of crossing alignments in a heldout set of 10K sentence pairs. For EN-JA Otedama and `Lader` achieved crossing score reductions of over 20 points. However, `Lader` performed slightly better than Otedama under the intrinsic metric. Our implementation includes a script, `crossing-score.py`, for evaluating crossing score on heldout data. This measure is useful for quickly evaluating different configurations of Otedama, as well as comparing it to other pre-ordering approaches.

Evaluation by MT Performance Table 2 shows BLEU scores for Otedama and `Lader` experiments. For English-Japanese translation, Otedama performed on par with the *Lader class* model. Both models significantly outperformed the baseline by 0.7 and 0.8 BLEU. The best model, *Lader full*, outperformed Otedama by 1.2 BLEU points. However, the ranking changes for the German-English experiment. Here, Otedama and *Lader full* were indistinguishable, and both systems significantly improved over the baseline. *Lader class* did not produce a significant improvement, showing the importance of syntactic information for this language pair.

While the `Lader` models were equally good or better in terms of BLEU, this came at the cost of speed. Table 2 lists running times for the different systems. The benchmark was conducted on 100 randomly selected sentences, running 10 threads in parallel. We include parsing time for *Otedama* and *Lader full* (*Lader class* does not require parsing). Otedama ran 4.5–10 times faster than `Lader`, making it more practical to use, especially on large data.

4. Conclusion

We have presented a fast, flexible open-source implementation of automatic rule-learning for source-side pre-ordering from dependency-annotated aligned parallel

system	BLEU		Seconds/sentence	
	EN-JA	DE-EN	EN-JA	DE-EN
Baseline	31.6	38.1		
Otedama	32.3*	38.8*	0.64	0.35
Lader class	32.4*	38.4	2.89 ($\times$4.5)	2.38 ($\times$6.9)
Lader full	33.5*+	38.6*	4.58 ($\times$7.1)	3.72 ($\times$10.7)

Table 2: BLEU scores and run times (including parsing, where necessary) for different pre-ordering tools. * indicates a statistically significant improvement over the baseline. + indicates a statistically significant improvement over Otedama.

training data. Our tool supports several external parser formats, and has shown promising results on the difficult task of patent translation. Compared to another open-source pre-ordering tool, we achieved a speedup of 4.5–10 while maintaining translation performance.

Bibliography

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

Collins, Michael, Philipp Koehn, and Ivona Kučerová. Clause Restructuring for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, Stroudsburg, PA, USA, 2005.

de Gispert, Adrià, Gonzalo Iglesias, and Bill Byrne. Fast and Accurate Preordering for SMT using Neural Networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Denver, Colorado, 2015. Association for Computational Linguistics.

Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. In *Proceedings of the ACL 2010 System Demonstrations*, Uppsala, Sweden, 2010.

Dyer, Chris, Jonathan H. Clark, Alon Lavie, and Noah A. Smith. Unsupervised Word Alignment with Arbitrary Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, 2011.

Genzel, Dmitriy. Automatically Learning Source-side Reordering Rules for Large Scale Machine Translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, Beijing, China, August 2010.

Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria, 2013.

Jehl, Laura, Adrià de Gispert, Mark Hopkins, and Bill Byrne. Source-side Preordering for Translation using Logistic Regression and Depth-first Branch-and-Bound Search. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the Joint Conference on Human Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computational Linguistics*, Edmonton, Alberta, Canada, 2003.

Lerner, Uri and Slav Petrov. Source-Side Classifier Preordering for Machine Translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, October 2013.

Nakagawa, Tetsuji. Efficient Top-Down BTG Parsing for Machine Translation Preordering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Beijing, China, 2015.

Neubig, Graham, Taro Watanabe, and Shinsuke Mori. Inducing a Discriminative Parser to Optimize Machine Translation Reordering. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, Stroudsburg, PA, USA, 2012.

Sennrich, Rico, Gerold Schneider, Martin Volk, and Martin Warin. A New Hybrid Dependency Parser for German. In *Proceedings of the GSCL-Conference*, Potsdam, Germany, 2009.

Simianer, Patrick, Stefan Riezler, and Chris Dyer. Joint feature selection in distributed stochastic learning for large-scale discriminative training in SMT. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, Jeju Island, South Korea, 2012.

Socher, Richard, John Bauer, Christopher D. Manning, and Ng Andrew Y. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria, 2013. Association for Computational Linguistics.

Utiyama, Masao and Hitoshi Isahara. A Japanese-English patent parallel corpus. In *Proceedings of MT summit XI*, Copenhagen, Denmark, 2007.

Wäschle, Katharina and Stefan Riezler. Analyzing Parallelism and Domain Similarities in the MAREC Patent Corpus. *Multidisciplinary Information Retrieval*, pages 12–27, 2012.

Address for correspondence:
Stefan Riezler
riezler@cl.uni-heidelberg.de
Computational Linguistics
Heidelberg University
Im Neuenheimer Feld 325
69120 Heidelberg, Germany