PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012

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`

ISSN 0032-6585

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012

CONTENTS

Articles

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 5–24

Simple and Efficient Model Filtering in Statistical Machine Translation

Juan Pino, Aurelien Waite, William Byrne

Department of Engineering, University of Cambridge, Cambridge, CB2 1PZ, U.K.

Abstract

Data availability and distributed computing techniques have allowed statistical machine translation (SMT) researchers to build larger models. However, decoders need to be able to retrieve information efficiently from these models to be able to translate an input sentence or a set of input sentences. We introduce an easy to implement and general purpose solution to tackle this problem: we store SMT models as a set of key-value pairs in an HFile. We apply this strategy to two specific tasks: test set hierarchical phrase-based rule filtering and n-gram count filtering for language model lattice rescoring. We compare our approach to alternative strategies and show that its trade offs in terms of speed, memory and simplicity are competitive.

1. Introduction

Current machine translation research is characterised by ever increasing amounts of data available for research. For example, Figure 1 shows that for the WMT machine translation workshop (Callison-Burch et al., 2012) French-English constrained track translation task, the English side of parallel data has increased from 13.8M tokens in 2006 to 945.1M tokens in 2012 and that available English monolingual data has increased from 27.5M tokens to 6841.1M tokens. Along with growing amounts of data, the use of more powerful computers and distributed computing models such as MapReduce (Dean and Ghemawat, 2008; Lin and Dyer, 2010) has enabled machine translation researchers to build larger statistical machine translation (SMT) models. Examples include language modelling (Brants et al., 2007), translation rule extraction (Dyer et al., 2008; Weese et al., 2011), word alignment (Dyer et al., 2008; Lin and Dyer,

Corresponding author: `jmp84@cam.ac.uk`

Cite as: Juan Pino, Aurelien Waite, William Byrne. Simple and Efficient Model Filtering in Statistical Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 5–24.
doi: 10.2478/v10108-012-0005-x.

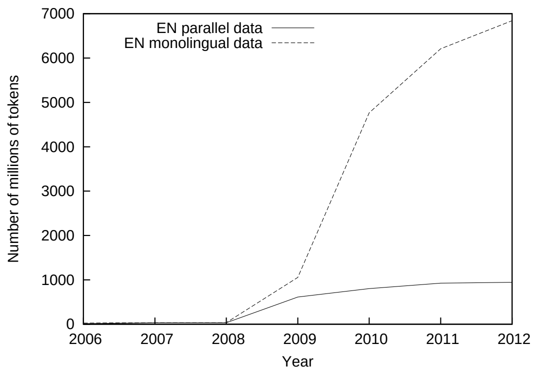


Figure 1. Number of English tokens (in millions) in parallel and monolingual data available for the WMT translation shared task constrained track for the years 2006 to 2012.

2010) as well as end-to-end toolkits for building entire phrase-based (Gao and Vogel, 2010) or hierarchical phrase-based models (Venugopal and Zollmann, 2009) using MapReduce.

Once SMT models are built, specifically the language model and the translation model, decoders or rescorers only need a fraction of the information contained in those models to be able to translate an input source sentence or a set of input source sentences. For example, in translation from French to English, given an input sentence "Salut toi", we don't need to know what translation probability the model assigns to other words than "Salut" and "toi" or what probability the English language model assigns to their possible English translation. With larger models, simply retrieving relevant translation or language model probabilities becomes a challenge. We use the HiFST system (Iglesias et al., 2009b; de Gispert et al., 2010), which involves a first-pass decoding followed by a 5-gram language model lattice rescoring step (Blackwood, 2010). Given a test set, the decoder only needs the rules whose source side matches part of one of the source sentences in the test set to be able to generate hypotheses. In the system described by Iglesias et al. (2009b), for each new test set, rules are re-extracted and filtered at extraction time. Similarly, for the task of 5-gram language model lattice rescoring (Blackwood, 2010), the rescorer only needs to retrieve counts for n-grams present in the lattice produced by the first-pass decoder to be able to assign a score to a hypothesis. As described by Blackwood (2010), obtaining relevant n-grams with their counts involves scanning a very large text file containing n-grams and counts and keeping the relevant records.

These two methods become progressively slower with larger amounts of data and we would like to improve on them for more rapid experimentation. We also would like to use as lightweight a computing infrastructure as possible. For example, HBase has been applied to the use of distributed language models (Yu, 2008). However, we wish to address the question whether we can adapt this heavy infrastructure to our purposes with minimal effort. N-gram count filtering and rule filtering are two essential steps in our pipeline that can be a bottleneck. Our goal is to reduce their processing time from several hours to a few minutes.

This paper addresses the problem of retrieving relevant translation and language model probabilities by storing models in the HFile data structure.[1] To our knowledge, this is the first detailed proposed implementation of translation and language model storage and filtering using HFile data structures. We believe it offers a good compromise between speed, performance and ease of implementation. Although the HFile construction is done via MapReduce and a cluster of machines, the infrastructure for filtering is lightweight and requires the use of only one machine. We will apply this approach for two specific tasks, namely test set rule filtering prior to decoding and n-gram count filtering to build a stupid backoff model (Brants et al., 2007) for lattice rescoring (Blackwood, 2010). We will discuss alternative strategies as well as their strengths and weaknesses in terms of speed and memory usage. In Section 2, we will review approaches that have been used for model filtering. The HFile data structure that is used to store models will be presented in Section 3. Our method and alternative strategies will be compared empirically in Sections 4 and 5. We will finally conclude in Section 6.

2. Related Work

We now review techniques appearing in the literature that have been used to store SMT models and to retrieve the information needed in translation from these models. SMT models are usually discrete probabilistic models and can therefore be represented as a set of key-value pairs. To obtain relevant information from a model stored in a data structure, a set of keys called a *query set* is formed, then each key in this query set is looked up in the model. Strategies include storing the model as a simple data structure in memory, in a plain text file, in more complicated data structures in memory, storing fractions of the entire model, simply storing data as opposed to a precomputed model or storing models in a distributed fashion.

If small enough, it may be possible to fit the model into physical memory. In this case the model can be stored as a memory associative array, such as a hash table, for rapid query retrieval. In-memory storage has been used to store model parameters between iterations of expectation-maximisation for word alignment (Dyer et al., 2008; Lin and Dyer, 2010).

For larger models, the set of key-value pairs can be stored as a table in a single text file on local disk. Values for keys in the query set are retrieved by scanning through

[1]http://hbase.apache.org

the entire file. For each key in the file, its membership is tested in the query set. This is the approach adopted in the *Joshua 3.0* decoder (Weese et al., 2011), which uses regular expressions or n-grams to test membership. Venugopal and Zollmann (2009) use MapReduce to scan a file concurrently: a mapper is defined that tests if the vocabulary of a rule matches the vocabulary of a test set. The MapReduce framework then splits the grammar file into subsections for the mappers to scan over in parallel.

The model can also be stored using a trie associative array (Fredkin, 1960). A trie is a type of tree where each node represents a shared prefix of a set of keys represented by the child nodes. Each node only stores the prefix it represents. The keys are therefore compactly encoded in the structure of the trie itself. Querying the trie is a $\mathcal{O}(\log(n))$ operation, where n is the number of keys in the dataset. The trie may also be small enough to fit in physical memory to further reduce querying time. Tries have been used for storing phrase tables (Zens and Ney, 2007) and hierarchical phrase-based grammars (Ganitkevitch et al., 2012) as well as language models (Pauls and Klein, 2011; Heafield, 2011).

It is also possible to create a much smaller approximate version of the model. Randomised language models (Talbot and Osborne, 2007b,a; Talbot and Brants, 2008) store parameters or counts associated with n-grams in a structure similar to a Bloom filter (Bloom, 1970). This structure is small in comparison to the original language model, although the reduction in size comes at the cost of randomly corrupting model parameters or assigning model parameters to unseen n-grams. Guthrie and Hepple (2010) propose an extension which prevents the random corruption of model parameters but does not stop the random assignment of parameters to unseen n-grams. Levenberg and Osborne (2009) extend randomised language models to stream-based language models. Another way of building a smaller approximate version of a model is to retain items with high frequency counts from a stream of data (Manku and Motwani, 2002). This technique has been applied to language modelling (Goyal et al., 2009) and translation rule extraction (Przywara and Bojar, 2011).

Instead of pre-computing the dataset it is possible to compute the sufficient statistics at query time using a suffix array (Manber and Myers, 1990), so that the model can be estimated on the fly. A suffix array is a sequence of pointers to each suffix in a training corpus. The sequence is sorted with respect to the lexicographic order of the referenced suffixes. Suffix arrays have been used for computing statistics for language models (Zhang and Vogel, 2006), phrase-based systems (Callison-Burch et al., 2005; Zhang and Vogel, 2005), and hierarchical phrase-based systems (Lopez, 2007).

Finally, some approaches store language models in a distributed fashion. Brants et al. (2007) describe a distributed, fast, low-latency infrastructure for storing very large language models. Zhang et al. (2006) propose a distributed large language model backed by suffix arrays. HBase has also been used to build a distributed language infrastructure (Yu, 2008). The method we propose to use is closely related to the latter but we use a more lightweight infrastructure than HBase and we apply it to two different tasks, demonstrating the flexibility of the infrastructure.

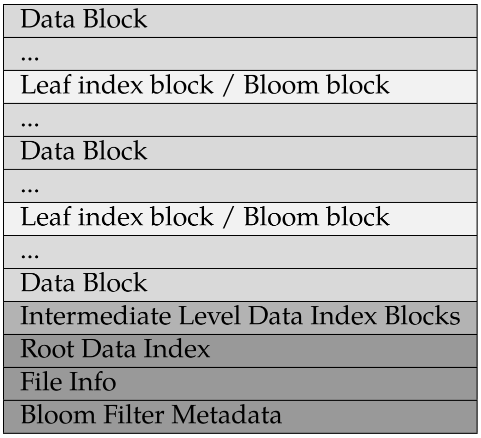

Data Block
...
Leaf index block / Bloom block
...
Data Block
...
Leaf index block / Bloom block
...
Data Block
Intermediate Level Data Index Blocks
Root Data Index
File Info
Bloom Filter Metadata

Figure 2. HFile internal structure [2]

3. HFile Description

We now describe the data structure we use to store models and we review relevant features to the design of our system. To store a model represented as key-value pairs, we use the HFile file format,[3] which is a reimplementation of the SSTable file format (Chang et al., 2008). The HFile is used at a lower level in the HBase infrastructure. In this work, we reuse the HFile format directly without having to install an HBase system. The HFile format is a lookup table with key and value columns. The entries are free to be an arbitrary string of bytes of any length. The table is sorted lexicographically by the key byte string for efficient record retrieval by key.

3.1. Internal structure

As can be seen in Figure 2, the data contained in an HFile is internally organised into blocks called data blocks. The block size is configurable, with a default size of 64KB. Note that HFile blocks are not to be confused with Hadoop Distributed File System (HDFS) blocks whose default size is 64MB. If an HFile is stored on HDFS, several HFile blocks will be contained in an HDFS block. A block index is constructed which maps the first key of an HFile block to the location of the block in the file. For large HFiles the block index can be very large. Therefore the block index is itself organised into blocks, which are called leaf index blocks. These leaf index blocks

[2]after http://hbase.apache.org/book/book.html (simplified)

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

are interspersed with the data blocks in the HFile. In turn, the leaf index blocks are indexed by intermediate level data index blocks. The intermediate blocks are then indexed by a root data index. The root data index and optionally the Bloom filter metadata, described next, are stored at the end of the HFile. In order to distinguish block types (data block, index block, etc.), the first 8 bytes of a block will indicate the type of block being read. The HFile format allows for the blocks to be compressed. The choice of compression codec is selected when the file is created. We choose the GZip compression codec for all our experiments. Block compression is also used in other related software (Pauls and Klein, 2011). For more details, the interested reader can refer to the HBase documentation.[4]

3.2. Record retrieval

When the HFile is opened for reading, the root data index is loaded into memory. To retrieve a value from the HFile given a key, the appropriate intermediate index block is located by a binary search through the root data index. Binary searches are conducted on the intermediate and leaf index blocks to identify the data block that contains the key. The data block is then loaded off the disk into memory and the key-value record is retrieved by scanning the data block sequentially.

3.3. Bloom filter optimization

It is possible to query for a key that is not contained in the HFile. This very frequently happens in translation because of language data sparsity. Querying the existence of a key is expensive as three blocks have to be loaded from disk and binary searched. For fast existence check queries, the HFile format allows the inclusion of an optional Bloom filter (Bloom, 1970). A Bloom filter provides a probabilistic, memory efficient representation of the key set with an $O(1)$ membership test operation. The Bloom filter may provide a false positive, but never a false negative for existence of a key in the HFile. For a large HFile, the Bloom filter may also be very large. Therefore the Bloom filter is also organised into blocks called Bloom blocks. Each block contains a smaller Bloom filter that covers a range of keys in the HFile. Similar to the root data index, a Bloom filter metadata or Bloom index is constructed. To check for the existence of a key, a binary search is conducted on the Bloom index, the relevant Bloom block is loaded, and the membership test performed. Contrary to work on Bloom filter language model (Talbot and Osborne, 2007a,b), this filter only tests the existence of a key and does not return any statistics from the value. If a membership test is positive, the HFile data structure still requires to do a usual search. During the execution of a query, two keys may reference the same index or Bloom blocks. To prevent these blocks from being repeatedly loaded from disk, they are cached after reading.

[4]http://hbase.apache.org/book/book.html

3.4. Local disk optimization

The HFile format is designed to be used with HDFS, a distributed file system based on the Google File System (Ghemawat et al., 2003). Large files are split into HDFS blocks that are stored on many nodes in a cluster. However, the HFile format can also be used completely independently of HDFS. If its size is smaller than disk space, the entire HFile can be stored on the local disk of one machine and accessed through the machine's local file system. We find in Sections 4 and 5 that using local disk is faster than using HDFS.

3.5. Query sorting optimization

Prior to HFile lookup, we sort keys in the query set lexicographically. If two keys in the set of queries are contained in the same block, then the block is only loaded once. In addition, the computer hardware and operating system allow further automatic improvements to the query execution. Examples of these automatic improvements include reduced disk seek time, the operating system caching data from disk,[5] or CPU caching data from main memory (Patterson and Hennessy, 2009).

4. Hierarchical Rule Filtering for Translation

In this section, we describe how the HFile data structure can be used to store a hierarchical phrase-based translation model (Chiang, 2007) and to retrieve rules from a given test set. We describe our system called *ruleXtract*, and compare it to other methods through time and memory measurements.

4.1. Task Description

Given a test set and a hierarchical phrase-based translation model, we would like to retrieve all the relevant rules from the model. A phrase-based rule is relevant if its source is a substring of a sentence in the test set. A hierarchical rule is relevant if, after instantiation of its nonterminals, it is a substring of a sentence in the test set. For example, with a test set containing one sentence "Salut toi", the phrase-based rules with sources "Salut", "toi", "Salut toi" are relevant and the hierarchical rules with sources "Salut X" and "X toi" are relevant.

4.2. HFile for Hierarchical Phrase-Based Grammars

The input to our system *ruleXtract* is a word aligned parallel corpus. First, hierarchical phrase-based rules are extracted using a MapReduce job with no reducer.

[5]The Linux Documentation Project, The File System, http://tldp.org

Then, features that require a pass over the whole training material, such as the source-to-target probability, are computed in parallel using MapReduce jobs. We call these features MapReduce features. We follow Method 3 described by Dyer et al. (2008) to compute translation probabilities. Finally, the outputs of the feature jobs are merged in sorted order and the merged output is converted to an HFile. This step is preferably run on a cluster of machines.

Given a test set and an HFile storing a hierarchical phrase-based grammar, we first generate queries from the test set, then retrieve relevant rules along with their MapReduce features from the HFile. To generate queries, we have a set of allowed *source patterns* and instantiate these patterns against the test set. A source pattern is simply a regular expression. For example, the pattern $\Sigma^{+}X$ represents a rule source side containing a sequence of terminals followed by the nonterminal X. If the input sentence is "Salut à toi", the pattern will be instantiated as "Salut X" and "Salut à X". We impose the following constraints on source pattern instantiation where the first three relate to constraints in extraction and the last one relates to a decoding constraint:

- *max_source_phrase*: maximum number of terminals for phrase-based rules,
- *max_source_elements*: maximum number of terminals and nonterminals,
- *max_terminal_length*: maximum number of consecutive terminals for hierarchical rules,
- *max_nonterminal_span*: maximum nonterminal span in a hierarchical rule.

The source pattern instances are then sorted for more efficient HFile lookup (see Section 3). Each query is then looked up in the HFile and if present, an HFile record is retrieved. We typically run this retrieval step on one machine only.

We now compare our approach to similar approaches whose aim is to obtain rules for a test set.

4.3. Suffix Array for Hierarchical Phrase-Based Grammars

We use the *cdec* software (Dyer et al., 2010) for hierarchical phrase-based rule extraction. The implementation is based on earlier work (Lopez, 2007) which extends suffix array based rule retrieval from phrase-based systems to hierarchical phrase-based systems.

Given a test set, a set of source pattern instances is generated similarly to what is done for *ruleXtract*. Then these source pattern instances are looked up in a suffix array compiled from the source side of a parallel corpus. Rules are then extracted using the word alignment and source-to-target probabilities are then computed on the fly.

4.4. Text File Representation of Hierarchical Phrase-Based Grammars

We now describe an implementation for storing and retrieving from a translation model by the *Joshua* decoder (Weese et al., 2011). The first implementation variant, which we call *Joshua*, stores the translation model in a text file. Given a test set, each word in the test set vocabulary is mapped to the list of sentences in which it appears. Then, each rule in the translation model is compiled to a regular expression, and each sentence that contains at least a vocabulary word of the rule is matched against this regular expression. If at least one match is successful, the rule is retained. A faster version is provided that matches consecutive terminals in the source side of a rule to the set of n-grams extracted from the test set. We call this version *Joshua Fast*. A parallel version also exists that chunks the grammar file and distributes each chunk processing as a separate process on a cluster running Sun Grid Engine (Gentzsch, 2001). We call this version *Joshua Parallel*. The parallel version using the faster matching algorithm is called *Joshua Fast Parallel*.

4.5. Experimental Setup

We use the following setup:

- Data: we use a small parallel corpus of 750,950 word-aligned sentence pairs and a larger corpus of 9,221,421 word-aligned sentence pairs from the NIST'12 Chinese-English evaluation, to show how systems scale up with more data.
- Grammar extraction: from the parallel corpora, we extract hierarchical grammars with the source-to-target probability feature only, because we do not want feature computation to introduce noise in timing results when comparing different strategies and software implementations. In addition, the suffix array implementation of rule extraction does not generate target-to-source probabilities. Note that in practice, given a vector of parameters, we could simply replace multiple features in the translation model by a single value representing the dot product of the features with the parameter vector. The extraction constraints are
  - max_source_phrase = 9,
  - max_source_elements = 5,
  - max_terminal_length = 5 (redundant with max_source_elements),
  - max_nonterminal_span = 10.

  The small grammars contains approximately 60M rules while the larger grammar contains approximately 726M rules. The grammar we obtain is converted to the *Joshua* format.
- Grammar filtering: for *ruleXtract*, we use these constraints for source pattern instantiation:
  - max_source_phrase = 9,
  - max_source_elements = 5,
  - max_terminal_length = 5 (redundant with max_source_elements),
  - max_nonterminal_span = ∞,

  so that the number of rules obtained after filtering is identical between *ruleXtract* and *Joshua*. For the *Joshua Parallel* configurations, we use 110 jobs for the larger grammar on a cluster of 9 machines. For this latter configuration, we report the maximum time spent on a job (not the sum) and the maximum memory usage by a job.

- Measurements: we report time measurements for query processing and query retrieval and the total time used to obtain a set specific rule file for a test set of 1755 Chinese sentences and 51008 tokens. We also report peak memory usage. For *ruleXtract*, query processing involves generating source pattern instances and sort them according to the HFile sorting order. If we use a Bloom filter, it also involves pre-filtering the queries with the Bloom filter. Query retrieval involves HFile lookup. For the *Joshua* configurations, query processing involves indexing the test set and generating test set ngrams and query retrieval involves regular expression matching.
- Hardware configuration: the machine used for the query has 94GB of memory and an Intel Xeon X5650 CPU. The distributed file system is hosted on the querying machine and other machines with the same specification, which are used to generate the HFile.

The setup was designed for accurate comparisons between strategies, however these strategies are not necessarily used with this setup in an end-to-end translation system. For example the grammar extracted by *Joshua* is smaller than the grammar extracted by *ruleXtract* because of target side constraints but *ruleXtract* uses filter criteria (Iglesias et al., 2009a) to reduce the test set specific grammar.

4.6. Results and Discussion

Results are summarized in Table 1, from which we can draw the following observations:

- Speed: column *Total Time* shows that *ruleXtract* is competitive with alternative strategies in terms of speed.
- Memory: column *Peak Memory* shows that both *ruleXtract* and *Joshua* memory usage is important. In the case of *ruleXtract*, this is because we keep all source pattern instances in memory. In the case of *Joshua*, this is due to a caching optimization.
- HFile optimization: comparing *HDFS* and *Local* rows, we can see that using the local filesystem as opposed to HDFS gives a small decrease in query retrieval time, more important for the larger grammar. This is due to the fact that HDFS blocks are located on different data nodes. Since the HFile size is smaller than the disk space, it is preferable to work locally, although it requires copying the HFile from HDFS to the hard disk. Comparing rows with and without *Bloom*, we can see that the use of a Bloom filter gives an important decrease in query retrieval time. This is due to the fact that the number of source pattern instances queries is 31,552,746 and after Bloom filtering, the number of queries is 1,146,554 for the small grammar and 2,309,680 for the larger grammar, reducing the num-

Small Grammar					
System	**Query Processing**	**Query Retrieval**	**Total Time**	**Peak Memory**	**# Rules**
ruleXtract HDFS	9m1s	7m36s	16m40s	40.8G	6435124
ruleXtract Bloom, HDFS	8m57s	2m16s	11m15s	39.9G	6435124
ruleXtract Local	8m54s	7m33s	16m30s	40.4G	6435124
ruleXtract Bloom, Local	8m50s	2m19s	11m11s	38.8G	6435124
Joshua	0.9s	29m51s	29m54s	42.2G	6435124
Joshua Fast	0.9s	7m25s	7m28s	40.1G	7493178
Large Grammar					
System	**Query Processing**	**Query Retrieval**	**Total Time**	**Peak Memory**	**# Rules**
ruleXtract HDFS	8m56s	22m18s	31m17s	42.2G	47978228
ruleXtract Bloom, HDFS	9m12	15m33s	24m49s	40.7G	47978228
ruleXtract Local	8m55s	21m3s	30m1s	41.6G	47978228
ruleXtract Bloom, Local	9m0s	14m43s	23m46s	40.6G	47978228
Joshua	0.9s	out of memory	out of memory	out of memory	out of memory
Joshua Fast	0.9s	out of memory	out of memory	out of memory	out of memory
Joshua No Cache	0.9s	537m10s	537m11s	10.1G	47978228
Joshua Fast No Cache	0.9s	78m53s	78m54s	10.1G	83339443
Joshua Parallel	total time (not sum): 43m36s			4G	47978228
Joshua Fast Parallel	total time (not sum): 44m29s			4G	83339443

Table 1. Time and memory measurements for rule filtering with different strategies for a small and a large grammar.

ber of time consuming HFile lookups respectively by 96% and 93%. Note that Bloom filters increase query processing time only in the case of a large grammar and more so when using HDFS.

- Parallelization: in order to run *Joshua* on the larger grammar and avoid memory problems, we needed to use parallelization, which provided competitive speeds and a low memory footprint (maximum 4G per job). We are currently looking into making a parallel version of ruleXtract by parallelizing the query.

For information, *cdec*'s total processing time is 57m40s for the small grammar, which is significantly slower than the other methods. However, we do not include a direct comparison to *cdec* in Table 1 because the suffix array method involves much on-the-fly computation that has been precomputed in the case of *Joshua* and *ruleXtract*. Despite this apparent slowness, the use of suffix array methods for rule extraction favors rapid experimentation because no precomputation is required. But we note that the HFile generation from the larger parallel corpus took 5 hours and from this HFile it is possible to run multiple experiments by varying test sets and/or filtering parameters.

The *ruleXtract* system works in batch mode and dividing the number of words in the test set by the total time in the best configuration (*ruleXtract, Bloom, Local*) for the large grammar yields a speed of 35.8 words per second which is a real time system speed for batch processing tasks in which latency has little effect. However, running the system in that configuration gives a speed of 2.5 words per second for the longest sentence in the test set (135 words) and 1.3 words per second for a sentence of length 20. Future work will be dedicated to reduce latency and obtain an actual real time system.

5. n-Gram Count Filtering for Language Model Lattice Rescoring

In this section, we describe an HFile based infrastructure that supports a stupid backoff (Brants et al., 2007) n-gram language model. We conduct timed queries with comparison to a suffix array baseline.

5.1. Task Description

Blackwood (2010) motivates the use of 5-gram language model rescoring as a way of avoiding memory limitations in language model estimation and decoding. Depending on translation grammar and language model complexity, pruning thresholds in search can be set so that search errors are inconsequential. 5-gram rescoring requires first to obtain n-gram counts for $n \leq 5$ for a large monolingual corpus. Given a test set, n-grams present in the output lattices generated by a first-pass decoder are then extracted. The stupid backoff n-gram model (Brants et al., 2007) is described with the pseudo-probability $S(\cdot)$. It has the form:

$$S(W_i|W_{i-n+1}^{i-1}) = \begin{cases} \frac{f(W_{i-n+1}^{i})}{f(W_{i-n+1}^{i-1})} & \text{if } f(W_{i-n+1}^{i}) > 0 \\ \alpha S(W_i|W_{i-n+2}^{i}) & \text{otherwise} \end{cases} \tag{1}$$

where W_i^j is a sequence of words contained in a machine translation hypothesis, $f(W_i^j)$ is the count of the occurrences of the word sequence W_i^j in a large monolingual corpus, and α is a constant that is set heuristically. To compute the pseudo-probability of an n-gram $S(W_i|W_{i-n+1}^{i-1})$ the only statistics required are the counts for the constituent word sequences extracted from the monolingual corpus. Brants et al. (2007) show that with large amounts of data, stupid backoff smoothing performs similarly to Kneser-Ney smoothing (Kneser and Ney, 1995).

5.2. HFile for n-gram count filtering

The HFile stores n-grams W_i^j as keys, and their counts $f(W_i^j)$ as values. Each word of the key is mapped to an integer so that the n-gram becomes a string of integers. Each integer is then converted into a binary representation with a three byte width, which is adequate for the vocabulary used by our collections. The count is stored using a four byte integer representation.

5.3. Suffix array

The suffix array baseline is a modified version of the Suffix Array Language Model toolkit (SALM) toolkit (Zhang and Vogel, 2006). The original SALM toolkit used a 32-bit integer representation for each element in the suffix array. This representation has been widened to 64-bits to allow a larger corpus to be indexed. SALM loads the suffix array and monolingual corpus into memory for fast computation of the counts.

5.4. Experimental setup

We use the following setup:

- Data: We use a concatenation of the Gigaword Fifth Edition (Parker et al., 2011) with the English side of the NIST'12 parallel data for the constrained track. The SALM toolkit imposes a 256 word limit on sentence length in the corpus, therefore we truncated all sentences to 256 words. The corpus contains 5.4 billion words. From the monolingual corpus we extract 2.5 billion word sequences and counts. These are stored in an HFile with 8 KB data block size.
- Translation lattices: we replicate an experiment where a set of 2816 translation lattices are rescored using a 5-gram stupid backoff language model (Blackwood, 2010). The n-gram keys required to build the set-specific language model are extracted from the lattices using modified counting transducers (Mohri, 2003).

The queries take the form of 8.4 million keys, of which 7.3 million of the keys are unique.
- HFile optimization: we execute four HFile based queries based on whether the HFile contains a Bloom filter index, and whether the HFile is stored on local disk or a distributed file system.
- Time measurement phases: we split the query execution into distinct phases. For SALM we record the time taken to load the suffix array and monolingual corpus into memory, which we label index load time. We then enumerate through the unsorted keys in the query and compute the count associated with the key. Note that for any duplicate key in the query a duplicate count is computed. We call this phase query retrieval. For the HFile based infrastructure, the query has to be sorted. A Bloom filter may also be applied after the sort. We call this phase query processing. We then look up the HFile to locate the query keys. The look up phase is also labelled query retrieval.
- Hardware configuration: identical to the one in Section 4.

5.5. Results and Discussion

The results are shown in Table 2 from which we can draw the following observations:
- Speed: column 4 shows that the HFile infrastructure provides a competitive query speed with respect to SALM.
- Memory: column 5 shows that the memory overhead of the HFile infrastructure is much lower than SALM. We could reduce the suffix array memory usage by doing an on-disk binary search but this would increase the query processing time.
- HFile optimizations: an interesting result is the effect that the Bloom filter has on the query processing time for the distributed query. The time spent loading the blocks that comprise the Bloom filter offsets the time saved retrieving the counts. However, when using local disk the Bloom filter has only a small impact on the query processing time.

In addition, although disk usage is not an issue, it is worth mentioning that the English monolingual data together with the suffix array represent 90G of uncompressed data and the HFile size is 11G without Bloom filter and 14G with Bloom filter. We store the English monolingual data in a decompressed file for more efficient loading into a suffix array. On the other hand, only HFile blocks potentially containing a key are uncompressed during an HFile query.

We did not report comparisons to the KenLM toolkit (Heafield, 2011), which is designed for retrieving n-gram probabilities from an ARPA file as opposed to raw n-gram counts. It might be possible to build an ARPA file containing n-gram counts; we leave this study to future work and hope to obtain improvements in speed.

	Index Load	Query Processing	Query Retrieval	Total Time	Peak Memory
Suffix Array	8m39s	-	3m20s	11m59s	90.7G
HFile, HDFS	-	18s	3m54s	4m12s	3.1G
HFile, Bloom, HDFS	-	1m11s	2m52s	4m3s	5.8G
HFile, Local	-	18s	3m5s	3m23s	3.1G
HFile, Bloom, Local	-	25s	1m56s	2m21s	5.8G

Table 2. Timing results for n-gram count queries.

6. Conclusion

We have presented a strategy to filter SMT models to a given test set. This strategy is easy to implement, flexible as it was applied to two different tasks and it does not require extensive computing resources as it is run on one machine. We have demonstrated that its performance in terms of speed and memory usage is competitive with other current alternative approaches.

In the future, we would like to provide two extensions to our HFile infrastructure. First, in order to increase the speed in batch mode, we would like to implement a MapReduce version that would split the queries (as opposed to the HFile). Second, in order to provide a real time system, we would like to reduce latency by optimizing the source pattern instance creation phase.

Acknowledgements

This research was funded by two EPSRC DTA studentships for the first two authors, the European Union Seventh Framework Programme (FP7-ICT-2009-4) under grant agreement number 247762 and the DARPA BOLT program. We thank Gonzalo Iglesias for providing advice on an earlier draft, Juri Ganitkevitch and Matt Post for their help with the Joshua software and Kenneth Heafield and Jonathan Clark for answering questions about the KenLM software.

Bibliography

Blackwood, Graeme. *Lattice Rescoring Methods for Statistical Machine Translation*. PhD thesis, University of Cambridge, 2010.

Bloom, Burton H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, July 1970. ISSN 0001-0782. doi: 10.1145/362686.362692.

Brants, Thorsten, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Callison-Burch, Chris, Colin Bannard, and Josh Schroeder. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 255–262, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219872.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. Findings of the 2012 workshop on statistical machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 10–51, Montréal, Canada, June 2012. Association for Computational Linguistics.

Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4:1–4:26, June 2008. doi: 10.1145/1365815.1365816.

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

de Gispert, Adrià, Gonzalo Iglesias, Graeme Blackwood, Eduardo R. Banga, and William Byrne. Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Computational Linguistics*, 36(3):505—533, 2010. ISSN 0891-2017.

Dean, Jeffrey and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492.

Dyer, Chris, Aaron Cordova, Alex Mont, and Jimmy Lin. Fast, easy, and cheap: Construction of statistical machine translation models with MapReduce. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 199–207, Columbus, Ohio, June 2008. 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*, pages 7–12, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Fredkin, Edward. Trie memory. *Communications of the ACM*, 3(9):490–499, September 1960. ISSN 0001-0782. doi: 10.1145/367390.367400.

Ganitkevitch, Juri, Yuan Cao, Jonathan Weese, Matt Post, and Chris Callison-Burch. Joshua 4.0: Packing, PRO, and paraphrases. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 283–291, Montréal, Canada, June 2012. Association for Computational Linguistics.

Gao, Qin and Stephan Vogel. Training phrase-based machine translation models on the cloud: Open source machine translation toolkit Chaski. *The Prague Bulletin of Mathematical Linguistics*, 93:37–46, January 2010. doi: 10.2478/v10108-010-0004-8.

Gentzsch, Wolfgang. Sun grid engine: Towards creating a compute power grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, CCGRID '01, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1010-8.

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi: 10.1145/945445.945450.

Goyal, Amit, Hal Daume III, and Suresh Venkatasubramanian. Streaming for large scale NLP: Language modeling. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 512–520, Boulder, Colorado, June 2009. Association for Computational Linguistics.

Guthrie, David and Mark Hepple. Storing the web in memory: Space efficient language models with constant time retrieval. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 262–272, Cambridge, MA, October 2010. Association for Computational Linguistics.

Heafield, Kenneth. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

Iglesias, Gonzalo, Adrià de Gispert, Eduardo R. Banga, and William Byrne. Rule filtering by pattern for efficient hierarchical translation. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 380–388, Athens, Greece, March 2009a. Association for Computational Linguistics.

Iglesias, Gonzalo, Adrià de Gispert, Eduardo R. Banga, and William Byrne. Hierarchical phrase-based translation with weighted finite state transducers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 433–441, Boulder, Colorado, June 2009b. Association for Computational Linguistics.

Kneser, Reinhard and Hermann Ney. Improved backing-off for m-gram language modeling. In *Proceedings of ICASSP*, volume 1, pages 181–184, 1995.

Levenberg, Abby and Miles Osborne. Stream-based randomised language models for SMT. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 756–764, Singapore, August 2009. Association for Computational Linguistics.

Lin, Jimmy and Chris Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool Publishers, 2010.

Lopez, Adam. Hierarchical phrase-based translation with suffix arrays. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 976–985, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Manber, Udi and Gene Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the first ACM-SIAM symposium on Discrete algorithms*, pages 319–327. Society for Industrial and Applied Mathematics, 1990.

Manku, Gurmeet Singh and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

Mohri, Mehryar. Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(6):957–982, 2003.

Parker, Robert, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition. In *Linguistic Data Consortium, Phildelphia*, 2011.

Patterson, David A. and John L. Hennessy. *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2009.

Pauls, Adam and Dan Klein. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 258–267, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Przywara, Česlav and Ondřej Bojar. eppex: Epochal phrase table extraction for statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, 96:89–98, 2011.

Talbot, David and Thorsten Brants. Randomized language models via perfect hash functions. In *Proceedings of ACL-08: HLT*, pages 505–513, Columbus, Ohio, June 2008. Association for Computational Linguistics.

Talbot, David and Miles Osborne. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 512–519, Prague, Czech Republic, June 2007a. Association for Computational Linguistics.

Talbot, David and Miles Osborne. Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 468–476, Prague, Czech Republic, June 2007b. Association for Computational Linguistics.

Venugopal, Ashish and Andreas Zollmann. Grammar based statistical MT on Hadoop: An end-to-end toolkit for large scale PSCFG based MT. *The Prague Bulletin of Mathematical Linguistics*, 91:67–77, January 2009. doi: 10.2478/v10108-009-0017-3.

Weese, Jonathan, Juri Ganitkevitch, Chris Callison-Burch, Matt Post, and Adam Lopez. Joshua 3.0: Syntax-based machine translation with the Thrax grammar extractor. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 478–484, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

Yu, Xiaoyang. Estimating language models using Hadoop and HBase. Master's thesis, University of Edinburgh, Edinburgh, 2008.

Zens, Richard and Hermann Ney. Efficient phrase-table representation for machine translation with applications to online MT and speech translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 492–499, Rochester, New York, April 2007. Association for Computational Linguistics.

Zhang, Ying and Stephan Vogel. An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proceedings of the 10th Conference of the European Association for Machine Translation (EAMT-05*, pages 294–301, 2005.

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

Zhang, Ying, Almut Silja Hildebrand, and Stephan Vogel. Distributed language modeling for n-best list re-ranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 216–223, Sydney, Australia, July 2006. Association for Computational Linguistics.

Address for correspondence:
Juan Pino
jmp84@cam.ac.uk
Department of Engineering
University of Cambridge
Cambridge
CB2 1PZ, U.K.

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 25–35

Appraise: an Open-Source Toolkit for Manual Evaluation of MT Output

Christian Federmann

DFKI Language Technology Lab

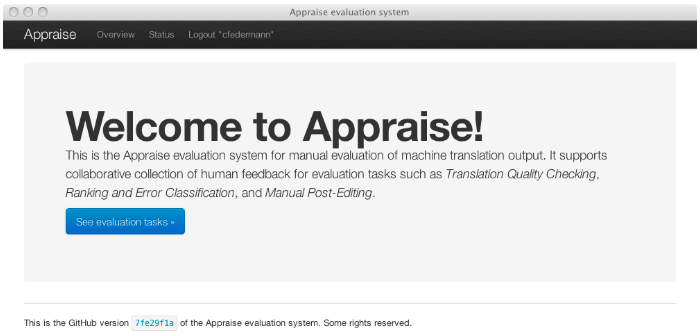
Abstract

We describe Appraise, an open-source toolkit supporting manual evaluation of machine translation output. The system allows to collect human judgments on translation output, implementing annotation tasks such as 1) quality checking, 2) translation ranking, 3) error classification, and 4) manual post-editing. It features an extensible, XML-based format for import/export and can easily be adapted to new annotation tasks. The current version of Appraise also includes automatic computation of inter-annotator agreements allowing quick access to evaluation results. Appraise is actively developed and used in several MT projects.

1. Introduction

Evaluation of Machine Translation (MT) output to assess translation quality is a difficult task. There exist automatic metrics such as BLEU (Papineni et al., 2002) or Meteor (Denkowski and Lavie, 2011) which are widely used in minimum error rate training (Och, 2003) for tuning of MT systems and as evaluation metric for shared tasks such as, e.g., the Workshop on Statistical Machine Translation (WMT) (Callison-Burch et al., 2012). The main problem in designing automatic quality metrics for MT is to achieve a high correlation with human judgments on the same translation output. While current metrics show promising performance in this respect, manual inspection and evaluation of MT results is still equally important as it allows for a more targeted and detailed analysis of the given translation output. The manual analysis of a given, machine translated text is a time-consuming and laborious process; it involves training of annotators, requires detailed and clear-cut annotation guidelines,

Corresponding author: cfedermann@dfki.de
Cite as: Christian Federmann. Appraise: an Open-Source Toolkit for Manual Evaluation of MT Output. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 25–35. doi: 10.2478/v10108-012-0006-9.


Appraise evaluation system

Appraise Overview Status Logout "cfedermann"

Welcome to Appraise!

This is the Appraise evaluation system for manual evaluation of machine translation output. It supports collaborative collection of human feedback for evaluation tasks such as *Translation Quality Checking*, *Ranking and Error Classification*, and *Manual Post-Editing*.

See evaluation tasks »

This is the GitHub version `7fe29f1a` of the Appraise evaluation system. Some rights reserved.

Figure 1. Front page

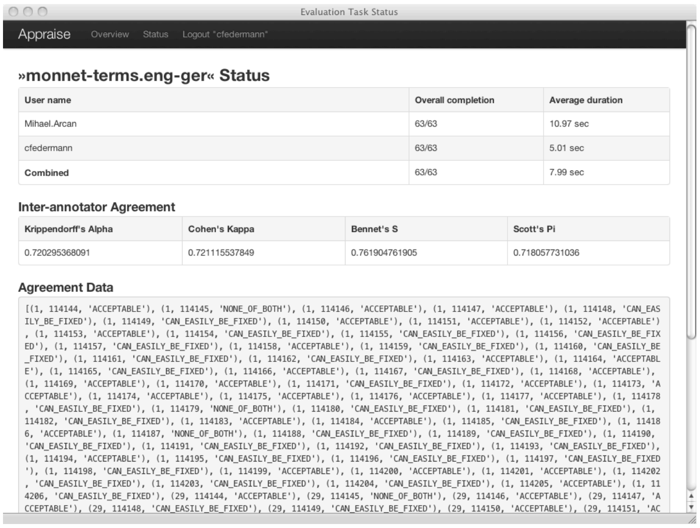
and—last but not least—an annotation software that allows annotators to get their job done quickly and efficiently.

In this paper, we describe *Appraise*, an open-source tool that allows to perform manual evaluation of Machine Translation output. Appraise can be used to collect human judgments on translation output, implementing several annotation tasks. We will describe the tool in more detail on the following pages. The remainder of this paper is structured as follows: Section 2 gives some further motivation concerning the development of the tool before we describe the system in more detail in Section 3 and highlight the various annotation tasks we implemented in Section 4. We explain the installation requirements in Section 5 and give some quick usage instructions in Section 6. Finally, we describe several experiments where Appraise has proven useful (see Section 7) and give some concluding remarks in Section 8.

2. Motivation

As we have mentioned before, the collection of manual judgments on machine translation output is a tedious task; this holds for simple tasks such as translation ranking but also for more complex challenges like word-level error analysis or post-editing of translation output. Annotators tend to lose focus after several sentences, resulting in reduced intra-annotator agreement and increased annotation time. In our experience with manual evaluation campaigns it has shown that a well-designed annotation tool can help to overcome these issues.

Development of the Appraise software package started back in 2009 as part of the EuroMatrixPlus project where the tool was used to quickly compare different sets of candidate translations from our hybrid machine translation engine to get an indication whether our system improved or degraded in terms of translation quality. A first version of Appraise was released and described by Federmann (2010).



»monnet-terms.eng-ger« Status

User name	Overall completion	Average duration
Mihael.Arcan	63/63	10.97 sec
cfedermann	63/63	5.01 sec
Combined	63/63	7.99 sec

Inter-annotator Agreement

Krippendorff's Alpha	Cohen's Kappa	Bennet's S	Scott's Pi
0.720295368091	0.721115537849	0.761904761905	0.718057731036

Agreement Data

```
[(1, 114144, 'ACCEPTABLE'), (1, 114145, 'NONE_OF_BOTH'), (1, 114146, 'ACCEPTABLE'), (1, 114147, 'ACCEPTABLE'), (1, 114148, 'CAN_EAS
ILY_BE_FIXED'), (1, 114149, 'CAN_EASILY_BE_FIXED'), (1, 114150, 'ACCEPTABLE'), (1, 114151, 'ACCEPTABLE'), (1, 114152, 'ACCEPTABLE')
, (1, 114153, 'ACCEPTABLE'), (1, 114154, 'CAN_EASILY_BE_FIXED'), (1, 114155, 'CAN_EASILY_BE_FIXED'), (1, 114156, 'CAN_EASILY_BE_FIX
ED'), (1, 114157, 'CAN_EASILY_BE_FIXED'), (1, 114158, 'ACCEPTABLE'), (1, 114159, 'CAN_EASILY_BE_FIXED'), (1, 114160, 'CAN_EASILY_BE
_FIXED'), (1, 114161, 'CAN_EASILY_BE_FIXED'), (1, 114162, 'CAN_EASILY_BE_FIXED'), (1, 114163, 'ACCEPTABLE'), (1, 114164, 'ACCEPTABL
E'), (1, 114165, 'CAN_EASILY_BE_FIXED'), (1, 114166, 'ACCEPTABLE'), (1, 114167, 'CAN_EASILY_BE_FIXED'), (1, 114168, 'ACCEPTABLE'),
(1, 114169, 'ACCEPTABLE'), (1, 114170, 'ACCEPTABLE'), (1, 114171, 'CAN_EASILY_BE_FIXED'), (1, 114172, 'ACCEPTABLE'), (1, 114173, 'A
CCEPTABLE'), (1, 114174, 'ACCEPTABLE'), (1, 114175, 'ACCEPTABLE'), (1, 114176, 'ACCEPTABLE'), (1, 114177, 'ACCEPTABLE'), (1, 114178
, 'CAN_EASILY_BE_FIXED'), (1, 114179, 'NONE_OF_BOTH'), (1, 114180, 'CAN_EASILY_BE_FIXED'), (1, 114181, 'CAN_EASILY_BE_FIXED'), (1,
114182, 'CAN_EASILY_BE_FIXED'), (1, 114183, 'ACCEPTABLE'), (1, 114184, 'ACCEPTABLE'), (1, 114185, 'CAN_EASILY_BE_FIXED'), (1, 11418
6, 'ACCEPTABLE'), (1, 114187, 'NONE_OF_BOTH'), (1, 114188, 'CAN_EASILY_BE_FIXED'), (1, 114189, 'CAN_EASILY_BE_FIXED'), (1, 114190,
'CAN_EASILY_BE_FIXED'), (1, 114191, 'CAN_EASILY_BE_FIXED'), (1, 114192, 'CAN_EASILY_BE_FIXED'), (1, 114193, 'CAN_EASILY_BE_FIXED'),
(1, 114194, 'ACCEPTABLE'), (1, 114195, 'CAN_EASILY_BE_FIXED'), (1, 114196, 'CAN_EASILY_BE_FIXED'), (1, 114197, 'CAN_EASILY_BE_FIXED
'), (1, 114198, 'CAN_EASILY_BE_FIXED'), (1, 114199, 'ACCEPTABLE'), (1, 114200, 'ACCEPTABLE'), (1, 114201, 'ACCEPTABLE'), (1, 114202
, 'CAN_EASILY_BE_FIXED'), (1, 114203, 'CAN_EASILY_BE_FIXED'), (1, 114204, 'CAN_EASILY_BE_FIXED'), (1, 114205, 'ACCEPTABLE'), (1, 11
4206, 'CAN_EASILY_BE_FIXED'), (29, 114144, 'ACCEPTABLE'), (29, 114145, 'NONE_OF_BOTH'), (29, 114146, 'ACCEPTABLE'), (29, 114147, 'A
CCEPTABLE'), (29, 114148, 'CAN_EASILY_BE_FIXED'), (29, 114149, 'CAN_EASILY_BE_FIXED'), (29, 114150, 'ACCEPTABLE'), (29, 114151, 'AC
```

Figure 2. Individual task status

3. System Description

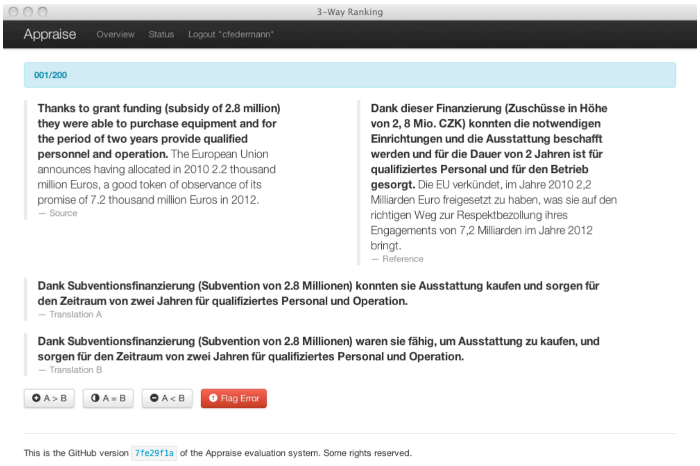
In a nutshell, Appraise is an open-source tool for manual evaluation of machine translation output. It allows to collect human judgments on given translation output, implementing annotation tasks such as (but not limited to):

- translation quality checking;
- ranking of translations;
- error classification;
- manual post-editing.

We will provide a more detailed discussion of these tasks in Section 4.

The software features an extensible XML import/output format and can easily be adapted to new annotation tasks. An example of this XML format is depicted in Figure 5. The software also includes automatic computation of inter-annotator agreement scores, allowing quick access to evaluation results. A screenshot of the task status view is shown in Figure 2. We currently support computation of the following inter-annotator agreement scores:

- Krippendorff's α as described by Krippendorff (2004);
- Fleiss' κ as published in Fleiss (1971), extending work from Cohen (1960);
- Bennett, Alpert, and Goldstein's S as defined in Bennett et al. (1954);
- Scott's π as introduced in Scott (1955).



001/200

Thanks to grant funding (subsidy of 2.8 million) they were able to purchase equipment and for the period of two years provide qualified personnel and operation. The European Union announces having allocated in 2010 2.2 thousand million Euros, a good token of observance of its promise of 7.2 thousand million Euros in 2012.
— Source

Dank dieser Finanzierung (Zuschüsse in Höhe von 2, 8 Mio. CZK) konnten die notwendigen Einrichtungen und die Ausstattung beschafft werden und für die Dauer von 2 Jahren ist für qualifiziertes Personal und für den Betrieb gesorgt. Die EU verkündet, im Jahre 2010 2,2 Milliarden Euro freigesetzt zu haben, was sie auf den richtigen Weg zur Respektbezollung ihres Engagements von 7,2 Milliarden im Jahre 2012 bringt.
— Reference

Dank Subventionsfinanzierung (Subvention von 2.8 Millionen) konnten sie Ausstattung kaufen und sorgen für den Zeitraum von zwei Jahren für qualifiziertes Personal und Operation.
— Translation A

Dank Subventionsfinanzierung (Subvention von 2.8 Millionen) waren sie fähig, um Ausstattung zu kaufen, und sorgen für den Zeitraum von zwei Jahren für qualifiziertes Personal und Operation.
— Translation B

This is the GitHub version `7fe29f1a` of the Appraise evaluation system. Some rights reserved.

Figure 3. 3-way ranking task

Agreement computation relies on code from the NLTK project (Bird et al., 2009). Additional agreement metrics can be added easily; the visualisation of agreement scores or other annotation results can be adapted to best match the corresponding annotation task design.

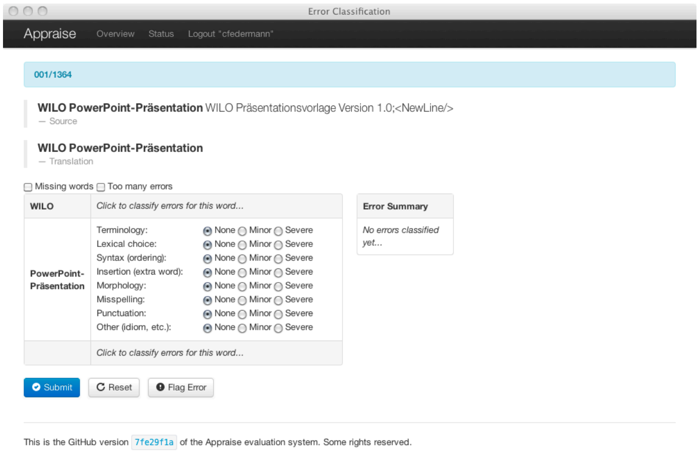
Appraise has been implemented using the Python-based *Django web framework*[1] which takes care of low-level tasks such as "HTTP handling", database modeling, and object-relational mapping. Figures 1–4 show several screenshots of the Appraise interface. We used Twitter's *Bootstrap*[2] as basis for the design of the application and implemented it using long-standing and well-established open-source software with large communities supporting them in the hope that this will also benefit the Appraise software package in the long run.

In the same spirit, we have opened up Appraise development and released the source code on GitHub at `https://github.com/cfedermann/Appraise`. Anybody with a free GitHub account may fork the project and create an own version of the software. Due to the flexibility of the `git` source code management system, it is easy to re-integrate external changes into the master repository, allowing other developers to feed back bug fixes and new features, thus improving and extending the original software. Appraise is available under an open, BSD-style license.[3]

[1]See `http://www.djangoproject.com/` for more information

[2]Available from `http://twitter.github.com/bootstrap/`

[3]See `https://raw.github.com/cfedermann/Appraise/master/appraise/LICENSE`



001/1364

WILO PowerPoint-Präsentation WILO Präsentationsvorlage Version 1.0;<NewLine/>
— Source

WILO PowerPoint-Präsentation
— Translation

☐ Missing words ☐ Too many errors

WILO	*Click to classify errors for this word...*
PowerPoint-Präsentation	Terminology: [✓ None] Minor Severe Lexical choice: [✓ None] Minor Severe Syntax (ordering): [✓ None] Minor Severe Insertion (extra word): [✓ None] Minor Severe Morphology: [✓ None] Minor Severe Misspelling: [✓ None] Minor Severe Punctuation: [✓ None] Minor Severe Other (idiom, etc.): [✓ None] Minor Severe
	Click to classify errors for this word...

Error Summary

No errors classified yet...

This is the GitHub version `7fe29f1a` of the Appraise evaluation system. Some rights reserved.

Figure 4. Error classification task

4. Annotation Tasks

We have developed several annotation tasks which are useful for MT evaluation. All of these have been tested and used during the experiments described in Section 7. The following task types are available for the GitHub version of Appraise:

1. **Ranking** The annotator is shown 1) the source sentence and 2) several ($n \geq 2$) candidate translations. It is also possible to additionally present the reference translation. Wherever available, one sentence of left/right context is displayed to support the annotator during the ranking process.

   We also have implemented a special *3-way ranking task* which works for pairs of candidate translations and gives the annotator an intuitive interface for quick $A > B$, $A = B$, or $A < B$ classification. Figure 3 shows a screenshot of the 3-way ranking interface.

2. **Error Classification** The annotator sees 1) the source (or target) sentence and 2) a candidate translation which has to be inspected wrt. errors contained in the translation output. We use a refined version of the classification described in (Vilar et al., 2006). Error annotation is possible on the sentence level as well as for individual words. The annotator can choose to skip translations containing "too many errors" and is able to differentiate between "minor" and "severe" errors. Figure 4 shows a screenshot of the error classification interface.

3. **Quality Estimation** The annotator is given 1) the source sentence and 2) one candidate translation which has to be classified as *Acceptable*, *Can easily be fixed*, or *None of both*. We also show the reference sentence and again present left/right context if available. This task can be used to get a quick estimate on the *acceptability* of a set of translations.

4. **Post-editing** The annotator is shown 1) the source sentence including left/right context wherever available and 2) one or several candidate translation. The task is defined as choosing the translation which is "easiest to post-edit" and then performing the post-editing operation on the selected translation.

   In some of our experiments with Appraise, we found that annotators did not necessarily choose the overall best candidate translation for post-editing but often selected worse translations which, however, could be post-edited more quickly. Our findings are summarised in Avramidis et al. (2012).

5. Installation Requirements

Appraise requires Python 2.7.x and Django 1.4.x to be installed on the deployment machine. You can install Python using the following commands:

```
$ wget http://www.python.org/ftp/python/2.7.3/Python-2.7.3.tgz
$ tar xzf Python-2.7.3.tgz
$ cd Python-2.7.3
$ ./configure && make && make install
```

After having set up Python, you have to download, extract, and install the Django web framework. This will be installed into the `site-packages` folder that belongs to the `python` binary used to start `setup.py`. Run the following commands:

```
$ wget djangoproject.com/download/1.4/tarball/ -O Django-1.4.1.tar.gz
$ tar xzvf Django-1.4.1.tar.gz
$ cd Django-1.4.1
$ python2.7 setup.py install
```

Note: on Mac OS X, you can also use MacPorts[4] to install Python and Django, simplifying the whole installation procedure to a single command:

```
$ sudo port install py27-django
```

Finally, you have to create a local copy of the Appraise source code package which is available from GitHub. In `git` terminology, you have to "clone" Appraise. You can do so as follows (change `Appraise-Software` to any other folder name you like):

[4]Available from `http://www.macports.org/`

```
$ git clone git://github.com/cfedermann/Appraise.git Appraise-Software
Cloning into 'Appraise-Software'...
...
$ cd Appraise-Software
```

Congratulations! You have just installed Appraise on your local machine.

6. Usage Instructions

Assuming you have already installed Python and Django, and have cloned a local copy of Appraise, you can setup the SQLite database and subsequently start up the server using the following commands:

```
$ cd Appraise-Software/appraise
$ python manage.py syncdb
...
```

When asked whether you want to create a super user account, reply `yes` and create such an account; this will be the administrative user having all permissions.

```
$ python manage.py runserver
Validating models...

0 errors found
Django version 1.4.1, using settings 'appraise.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

You should be greeted with the output shown above in your terminal. In case of any errors during startup, these will be reported instead and, depending on the severity of the problem, Django will refuse to launch Appraise. Point your browser to `http://127.0.0.1:8000/appraise/` and check if you can see the Appraise front page, which looks similar to the screenshot depicted in Figure 1.

New user accounts can be created inside Django's administration backend. You have to login and access `http://127.0.0.1:8000/appraise/admin/auth/user/add/` for user administration. Evaluation tasks are created in the administration backend at `http://127.0.0.1:8000/appraise/admin/evaluation/evaluationtask/add/`. You need an XML file in proper format to upload a task; an example file can be found inside `examples/sample-ranking-task.xml` within the Appraise package.

7. Experiments

7.1. Appraise in EuroMatrixPlus

As mentioned earlier in this article, we have created Appraise to support research work on hybrid machine translation, especially during the EuroMatrixPlus project. This is described in (Federmann et al., 2009, 2010; Federmann and Hunsicker, 2011; Hunsicker et al., 2012).

7.2. Appraise in taraXÜ

We have also used Appraise in the taraXÜ project, conducting several large annotation campaigns involving professional translators and language service providers. Results from this research work are summarised in (Avramidis et al., 2012).

7.3. Appraise in T4ME

In the T4ME project, we investigate how hybrid machine translation can be changed towards optimal selection from the given candidate translations. Part of the experimental setup is a shared task (ML4HMT) in which participants have to implement this optimal choice step. We used Appraise to assess the translation quality of the resulting systems. This is described in (Federmann, 2011; Federmann et al., 2012a,b).

Appraise has also been used in research related to the creation of standalone hybrid machine translation approaches. Related work is published as (Federmann, 2012).

7.4. Appraise in MONNET

We also used Appraise in the context of terminology translation for the business domain. These experiments are conducted as part of the MONNET project and are presented in (Arcan et al., 2012).

8. Conclusion and Outlook

We have described Appraise, an open-source tool for manual evaluation of machine translation output, implementing various annotation tasks such as ranking or error classification. We provided detailed instructions on the installation and setup of the tool and gave some brief introduction to its usage. Also, we reported on research work for which different versions of Appraise have been used, feeding back into the tool's development.

Maintenance and development efforts of the Appraise software package are ongoing. By publicly releasing the tool on GitHub, we hope to attract both new users and new developers to further extend and improve it. Future modifications will focus on new annotation tasks and a more accessible administration structure for large numbers of tasks. Last but not least, we intend to incorporate detailed visualisation of annotation results into Appraise.

Acknowledgements

This work has been funded under the Seventh Framework Programme for Research and Technological Development of the European Commission (grant agreement no.: 249119) through the T4ME contract. We are grateful to the anonymous reviewers for their valuable feedback on the draft version of this paper.

```
<set id="spiegel-20120210" source-language="ger" target-language="eng">
    <seg id="1" doc-id="source-text.de.txt">
      <source>In der syrischen Stadt Aleppo sind nach staatlichen Angaben
           mehrere grosse Sprengsätze detoniert, offenbar vor zwei Einrichtungen
            der Sicherheitskräfte.</source>
      <translation system="google">In the Syrian city of Aleppo after
           government data several large bombs are detonated, apparently, two
           institutions of the security forces.</translation>
      <translation system="bing">In Aleppo, Syria, Syrian several large
           explosive devices are detonates according to State, apparently before
            two installations of the security forces.</translation>
      <translation system="yahoo">In the Syrian city Aleppo detonated according
            to national instructions several large explosive devices, obviously
           before two mechanisms of the security forces.</translation>
    </seg>
    ...
</set>
```

Figure 5. Excerpt of sample import XML for an Appraise ranking task. For consistency and ease of use, the same format is used for all annotation tasks. The full file is available as `examples/sample-ranking-task.xml` *from the Appraise software package.*

Bibliography

Arcan, Mihael, Christian Federmann, and Paul Buitelaar. Using Domain-specific and Collaborative Resources for Term Translation. In *In Proceedings of the Sixth workshop on Syntax, Structure and Semantics in Statistical Translation*, Jeju, South Korea, July 2012. Association for Computational Linguistics (ACL).

Avramidis, Eleftherios, Aljoscha Burchardt, Christian Federmann, Maja Popovic, Cindy Tscherwinka, and David Vilar Torres. Involving Language Professionals in the Evaluation of Machine Translation. In *8th ELRA Conference on Language Resources and Evaluation*. European Language Resources Association (ELRA), 2012.

Bennett, E. M., R. Alpert, and A. C. Goldstein. Communications Through Limited-response Questioning. *Public Opinion Quarterly*, 18(3):303–308, 1954. doi: 10.1086/266520.

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, Beijing, 2009. ISBN 978-0-596-51649-9. doi: http://my.safaribooksonline.com/9780596516499. URL `http://www.nltk.org/book`.

Bojar, Ondrej, Miloš Ercegovčević, Martin Popel, and Omar Zaidan. A Grain of Salt for the WMT Manual Evaluation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 1–11, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W11-2101`.

Callison-Burch, Chris, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. Further Meta-Evaluation of Machine Translation. In *Proceedings of the Third Workshop on*

Statistical Machine Translation, pages 70–106, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W08/W08-0309`.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia, editors. *Proceedings of the Seventh Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Montréal, Canada, June 2012. URL `http://www.aclweb.org/anthology/W12-31`.

Cohen, J. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. ISSN 0013-1644.

Denkowski, Michael and Alon Lavie. Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 85–91, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology-new/W/W11/W11-2107`.

Federmann, Christian. Appraise: An Open-Source Toolkit for Manual Phrase-Based Evaluation of Translations. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valetta, Malta, May 2010. URL `http://www.lrec-conf.org/proceedings/lrec2010/pdf/197_Paper.pdf`.

Federmann, Christian. Results from the ML4HMT Shared Task on Applying Machine Learning Techniques to Optimise the Division of Labour in Hybrid Machine Translation. In *Proceedings of the International Workshop on Using Linguistic Information for Hybrid Machine Translation (LIHMT 2011) and of the Shared Task on Applying Machine Learning Techniques to Optimise the Division of Labour in Hybrid Machine Translation (ML4*. META-NET, 11 2011.

Federmann, Christian. Can Machine Learning Algorithms Improve Phrase Selection in Hybrid Machine Translation? In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 113–118. Association for Computational Linguistics (ACL), European Chapter of the Association for Computational Linguistics (EACL), 4 2012.

Federmann, Christian and Sabine Hunsicker. Stochastic Parse Tree Selection for an Existing RBMT System. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 351–357, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W11-2141`.

Federmann, Christian, Silke Theison, Andreas Eisele, Hans Uszkoreit, Yu Chen, Michael Jellinghaus, and Sabine Hunsicker. Translation Combination using Factored Word Substitution. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 70–74, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0x11`.

Federmann, Christian, Andreas Eisele, Yu Chen, Sabine Hunsicker, Jia Xu, and Hans Uszkoreit. Further Experiments with Shallow Hybrid MT Systems. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 77–81, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W10-1708`.

Federmann, Christian, Eleftherios Avramidis, Marta R. Costa-jussa, Josef van Genabith, Maite Melero, and Pavel Pecina. The ML4HMT Workshop on Optimising the Division of Labour in Hybrid Machine Translation. In *8th ELRA Conference on Language Resources and Evaluation*. European Language Resources Association (ELRA), 5 2012a.

Federmann, Christian, Maite Melero, and Josef van Genabith. Towards Optimal Choice Selection for Improved Hybrid Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, 97:5–22, 4 2012b.

Fleiss, J.L. Measuring Nominal Scale Agreement among Many Raters. *Psychological Bulletin*, 76 (5):378–382, 1971.

Hunsicker, Sabine, Yu Chen, and Christian Federmann. Machine Learning for Hybrid Machine Translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 312–316, Montréal, Canada, June 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-3138`.

Krippendorff, Klaus. Reliability in Content Analysis. Some Common Misconceptions and Recommendations. *Human Communication Research*, 30(3):411–433, 2004.

Och, Franz Josef. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1075096.1075117.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. URL `http://acl.ldc.upenn.edu/P/P02/P02-1040.pdf`.

Scott, William A. Reliability of Content Analysis: The Case of Nominal Scale Coding. *The Public Opinion Quarterly*, 19(3):321–325, 1955.

Vilar, David, Jia Xu, Luis Fernando D'Haro, and Hermann Ney. Error Analysis of Machine Translation Output. In *International Conference on Language Resources and Evaluation*, pages 697–702, Genoa, Italy, may 2006.

Address for correspondence:
Christian Federmann
`cfedermann@dfki.de`
DFKI Gmbh—Language Technology Lab
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 37–50

Hierarchical Phrase-Based Translation with Jane 2

Matthias Huck, Jan-Thorsten Peter, Markus Freitag, Stephan Peitz, Hermann Ney

Human Language Technology and Pattern Recognition Group, RWTH Aachen University

Abstract

In this paper, we give a survey of several recent extensions to hierarchical phrase-based machine translation that have been implemented in version 2 of Jane, RWTH's open source statistical machine translation toolkit. We focus on the following techniques: Insertion and deletion models, lexical scoring variants, reordering extensions with non-lexicalized reordering rules and with a discriminative lexicalized reordering model, and soft string-to-dependency hierarchical machine translation. We describe the fundamentals of each of these techniques and present experimental results obtained with Jane 2 to confirm their usefulness in state-of-the-art hierarchical phrase-based translation (HPBT).

1. Introduction

Jane (Vilar et al., 2010a) is an open source translation toolkit which has been developed at RWTH Aachen University and is freely available for non-commercial use. Jane provides efficient C++ implementations for hierarchical phrase extraction, optimization of log-linear feature weights, and parsing-based search algorithms. A modular design and flexible extension mechanisms allow for easy integration of novel features and translation approaches.

In hierarchical phrase-based translation (Chiang, 2005, 2007), a weighted synchronous context-free grammar is induced from parallel text. In addition to contiguous *lexical* phrases, *hierarchical* phrases with usually up to two gaps are extracted. Hierarchical decoding is carried out with a search procedure which is based on CYK+ parsing (Chappelier and Rajman, 1998). Standard features that are typically inte-

Corresponding author: huck@cs.rwth-aachen.de

Cite as: Matthias Huck, Jan-Thorsten Peter, Markus Freitag, Stephan Peitz, Hermann Ney. Hierarchical Phrase-Based Translation with Jane 2. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 37–50. doi: 10.2478/v10108-012-0007-8.

grated into hierarchical baseline setups are: phrase translation probabilities and lexical smoothing probabilities, each in both source-to-target and target-to-source translation directions, word and phrase penalty, binary features marking hierarchical phrases, glue rule, and rules with non-terminals at the boundaries, and an n-gram language model. Other common and simple features are source-to-target and target-to-source phrase length ratios and binary features marking phrases that have been seen more than a certain number of times—one, two, three or five times, for instance—in the training data.

Jane additionaly implements a number of advanced techniques. These range from discriminative word lexicon (DWL models and triplet lexicon models (Mauser et al., 2009; Huck et al., 2010) over syntactic enhancements like parse matching (Vilar et al., 2008), preference grammars (Venugopal et al., 2009; Stein et al., 2010) and pseudo-syntactic enhancements like poor man's syntax (Vilar et al., 2010b) to a variety of search strategies with diverse pruning approaches and language model (LM) score estimation heuristics (Huang and Chiang, 2007; Vilar and Ney, 2009, 2011). Log-linear parameter weights can be optimized with either the downhill simplex algorithm (Nelder and Mead, 1965), Och's minimum error rate training (MERT) (Och, 2003), or the margin infused relaxed algorithm (MIRA) (Chiang et al., 2009).

The purpose of this paper is to present some features that have been added to Jane in version 2, namely insertion and deletion models (Section 2), lexical scoring variants (Section 3), reordering extensions (Section 4), and soft string-to-dependency features (Section 5). We will not address Jane's basic functionality or any other non-standard techniques that are available in Jane. Many of them have been discussed in depth in previous publications (Stein et al., 2011; Vilar et al., 2012). We refer the reader to those and to the manual included in the Jane package. Advice on how to employ most of the features implemented in Jane can likewise be found in the manual. Jane 2 is available for download at `http://www.hltpr.rwth-aachen.de/jane/`.

1.1. Notational Conventions

In hierarchical phrase-based translation, we deal with rules $X \rightarrow \langle \alpha, \beta, \sim \rangle$ where $\langle \alpha, \beta \rangle$ is a bilingual phrase pair that may contain symbols from a non-terminal set, i.e. $\alpha \in (\mathcal{N} \cup V_F)^+$ and $\beta \in (\mathcal{N} \cup V_E)^+$, where V_F and V_E are the source and target vocabulary, respectively, and $\mathcal{N}$ is a non-terminal set which is shared by source and target. The left-hand side of the rule is a non-terminal symbol $X \in \mathcal{N}$, and the $\sim$ relation denotes a one-to-one correspondence between the non-terminals in α and in β. Let J_α denote the number of terminal symbols in α and I_β the number of terminal symbols in β. Indexing α with j, i.e. the symbol α_j, $1 \leq j \leq J_\alpha$, denotes the j-th terminal symbol on the source side of the phrase pair $\langle \alpha, \beta \rangle$, and analogous with β_i, $1 \leq i \leq I_\beta$, on the target side.

2. Insertion and Deletion Models

Insertion and deletion models are designed as a means to avoid the omission of content words in the hypotheses. In our case, they are implemented as phrase-level feature functions which count the number of inserted or deleted words (Huck and Ney, 2012). An English word is considered inserted or deleted based on lexical probabilities with the words on the foreign language side of the phrase. Lexical translation probabilities from different types of lexicon models may be employed within the insertion and deletion scoring functions, e.g. a model which is extracted from word-aligned training data and—given the word alignment matrix—relies on pure relative frequencies (henceforth denoted as *RF word lexicon*) (Koehn et al., 2003), or the IBM model 1 lexicon (henceforth denoted as *IBM-1*) (Brown et al., 1993).

We define insertion and deletion models, each in both source-to-target and target-to-source direction, by giving phrase-level scoring functions for the features. In the Jane 2 implementation, the feature values are precomputed and written to the phrase table. The features are then incorporated directly into the log-linear model combination of the decoder.

2.1. Insertion Models

The insertion model in source-to-target direction $t_{\mathrm{s2tIns}}(\cdot)$ counts the number of inserted words on the target side β of a hierarchical rule with respect to the source side α of the rule:

$$t_{\mathrm{s2tIns}}(\alpha, \beta) = \sum_{i=1}^{I_\beta} \prod_{j=1}^{J_\alpha} \left[p(\beta_i|\alpha_j) < \tau_{\alpha_j} \right] \tag{1}$$

Here, $[\cdot]$ denotes a true or false statement: The result is 1 if the condition is true and 0 if the condition is false. The model considers an occurrence of a target word e an insertion iff no source word f exists within the phrase where the lexical translation probability $p(e|f)$ is greater than a corresponding threshold τ_f.

In an analogous manner to the source-to-target direction, the insertion model in target-to-source direction $t_{\mathrm{t2sIns}}(\cdot)$ counts the number of inserted words on the source side α of a hierarchical rule with respect to the target side β of the rule:

$$t_{\mathrm{t2sIns}}(\alpha, \beta) = \sum_{j=1}^{J_\alpha} \prod_{i=1}^{I_\beta} \left[p(\alpha_j|\beta_i) < \tau_{\beta_i} \right] \tag{2}$$

Target-to-source lexical translation probabilities $p(f|e)$ are thresholded with values τ_e which may be distinct for each target word e. The model considers an occurrence of a source word f an insertion iff no target word e exists within the phrase with $p(f|e)$ greater than or equal to τ_e.

2.2. Deletion Models

The deletion model in source-to-target direction $t_{\text{s2tDel}}(\cdot)$ counts the number of deleted words on the source side α of a hierarchical rule with respect to the target side β of the rule:

$$t_{\text{s2tDel}}(\alpha, \beta) = \sum_{j=1}^{J_\alpha} \prod_{i=1}^{I_\beta} \left[p(\beta_i|\alpha_j) < \tau_{\alpha_j}\right] \tag{3}$$

It considers an occurrence of a source word f a deletion iff no target word e exists within the phrase with $p(e|f)$ greater than or equal to τ_f.

The target-to-source deletion model $t_{\text{t2sDel}}(\cdot)$ correspondingly considers an occurrence of a target word e a deletion iff no source word f exists within the phrase with $p(f|e)$ greater than or equal to τ_e:

$$t_{\text{t2sDel}}(\alpha, \beta) = \sum_{i=1}^{I_\beta} \prod_{j=1}^{J_\alpha} [p(\alpha_j|\beta_i) < \tau_{\beta_i}] \tag{4}$$

2.3. Thresholding Methods for Insertion and Deletion Models

We introduce thresholding methods for insertion and deletion models which set thresholds based on the characteristics of the lexicon model that is applied. We restrict ourselves to the description of the source-to-target direction.

individual τ_f is a distinct value for each f, computed as the arithmetic average of all entries $p(e|f)$ of any e with the given f in the lexicon model.

global The same value $\tau_f = \tau$ is used for all f. We compute this global threshold by averaging over the individual thresholds.

histogram n τ_f is a distinct value for each f. τ_f is set to the value of the $n+1$-th largest probability $p(e|f)$ of any e with the given f.

all All entries with probabilities larger than the floor value are not thresholded. This variant may be considered as *histogram* ∞.

median τ_f is a median-based distinct value for each f, i.e. it is set to the value that separates the higher half of the entries from the lower half of the entries $p(e|f)$ for the given f.

3. Lexical Scoring

Lexical scoring on phrase level is the standard technique for phrase table smoothing in statistical machine translation (Koehn et al., 2003; Zens and Ney, 2004). Jane 2 supports lexical smoothing as well as source-to-target sentence level lexical scoring within search with many types of lexicon models (Huck et al., 2011). Phrase-level lexical scores do not have to be calculated on demand for each hypothesis expansion, but can again be precomputed in advance and written to the phrase table. We present four scoring variants for lexical smoothing with RF word lexicons or IBM-1 which are provided by Jane 2. We describe the source-to-target directions. The target-to-source scores are computed similarly.

3.1. Phrase-Level Scoring Variants

The first scoring variant $t_{\mathrm{Norm}}(\cdot)$ uses an IBM-1 or RF lexicon model $p(e|f)$ to rate the quality of a target side β given the source side α of a phrase with an included length normalization:

$$t_{\mathrm{Norm}}(\alpha, \beta) = \sum_{i=1}^{I_\beta} \log \left(\frac{p(\beta_i|\mathrm{NULL}) + \sum_{j=1}^{J_\alpha} p(\beta_i|\alpha_j))}{1 + J_\alpha} \right) \tag{5}$$

By dropping the length normalization we arrive at the second variant $t_{\mathrm{NoNorm}}(\cdot)$:

$$t_{\mathrm{NoNorm}}(\alpha, \beta) = \sum_{i=1}^{I_\beta} \log \left(p(\beta_i|\mathrm{NULL}) + \sum_{j=1}^{J_\alpha} p(\beta_i|\alpha_j)) \right) \tag{6}$$

The third scoring variant $t_{\mathrm{NoisyOr}}(\cdot)$ is the noisy-or model proposed by Zens and Ney (Zens and Ney, 2004):

$$t_{\mathrm{NoisyOr}}(\alpha, \beta) = \sum_{i=1}^{I_\beta} \log \left(1 - \prod_{j=1}^{J_\alpha} (1 - p(\beta_i|\alpha_j)) \right) \tag{7}$$

The fourth scoring variant $t_{\mathrm{Moses}}(\cdot)$ is due to Koehn, Och and Marcu (Koehn et al., 2003) and is the standard method in the open-source Moses system (Koehn et al., 2007):

$$t_{\mathrm{Moses}}(\alpha, \beta, \{a_{ij}\}) = \sum_{i=1}^{I_\beta} \log \left(\begin{cases} \frac{1}{|\{a_i\}|} \sum_{j \in \{a_i\}} p(\beta_i|\alpha_j)) & \text{if} \quad |\{a_i\}| > 0 \\ p(\beta_i|\mathrm{NULL}) & \text{otherwise} \end{cases} \right) \tag{8}$$

This last variant requires the availability of word alignments $\{a_{ij}\}$ for phrase pairs $\langle \alpha, \beta \rangle$. We store the most frequent alignment during phrase extraction and use it to compute $t_{\mathrm{Moses}}(\cdot)$.

Note that all of these scoring methods generalize to hierarchical phrase pairs which may be only partially lexicalized. Unseen events are scored with a small floor value.

Source-to-target sentence-level scores are calculated analogous to Eq. (5), but with the difference that the quality of the target side β of a rule currently chosen to expand a partial hypothesis is rated given the whole input sentence f_1^J instead of the source side α of the rule only.

4. Reordering Extensions

In hierarchical phrase-based machine translation, reordering is modeled implicitely as part of the translation model. Hierarchical phrase-based decoders conduct phrase reorderings based on the one-to-one relation between the non-terminals on source and target side within hierarchical translation rules. Recently, some authors have been able to improve translation quality by augmenting the hierarchical grammar with more flexible reordering mechanisms based on additional non-lexicalized reordering rules (He et al., 2010b; Sankaran and Sarkar, 2012; Li et al., 2012). Extensions with lexicalized reordering models have also been presented in the literature lately (He et al., 2010b,a).

Jane 2 offers both the facility to incorporate grammar-based mechanisms to perform reorderings that do not result from the application of hierarchical rules (Vilar et al., 2010a) and the optional integration of a discriminative lexicalized reordering model (Zens and Ney, 2006; Huck et al., 2012). Jane 2 furthermore enables the computation of distance-based distortion costs.

4.1. Non-Lexicalized Reordering Rules

In order to allow for a more flexible arrangement of phrases in the hypotheses, a single swap rule

$$X \to \left\langle X^{\sim 0} X^{\sim 1}, X^{\sim 1} X^{\sim 0} \right\rangle \tag{9}$$

may be added supplementary to the standard initial rule and glue rule. The swap rule enables adjacent phrases to be transposed.

Other, more complex modifications to the grammar outright replace the standard initial rule and glue rule and implement jumps across blocks of symbols. Specific jump rules put jumps across blocks on source side into effect. Blocks that are skipped by the jump rules are translated without further jumps. Reordering within these windows is possible with hierarchical rules only.

4.2. Discriminative Lexicalized Reordering Model

The discriminative lexicalized reordering model (*discrim. RO*) tries to predict the orientation of neighboring blocks. We use two orientation classes *left* and *right*, in the same manner as described by Zens and Ney (2006). The reordering model is applied at the phrase boundaries only, where words which are adjacent to gaps within hierarchical phrases are defined as boundary words as well. The orientation probability is modeled in a maximum entropy framework (Berger et al., 1996). The feature set of the model may consist of binary features based on the source word at the current source position, on the word class at the current source position, on the target word at the current target position, and on the word class at the current target position. The reordering model is trained with the generalized iterative scaling (GIS)

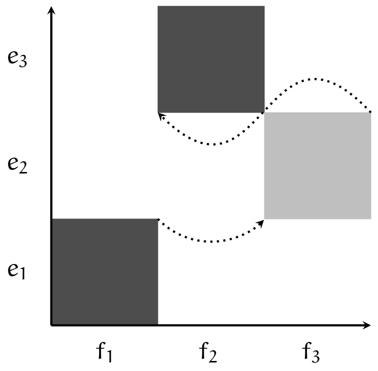


Figure 1. Illustration of an embedding of a lexical phrase (light) in a hierarchical phrase (dark), with orientations scored with the neighboring blocks.

algorithm (Darroch and Ratcliff, 1972) with the maximum class posterior probability as training criterion, and it is smoothed with a gaussian prior (Chen and Rosenfeld, 1999).

For each rule application during hierarchical decoding, the reordering model is applied at all boundaries where lexical blocks are placed side by side within the partial hypothesis. For this purpose, we need to access neighboring boundary words and their aligned source words and source positions. Note that, as hierarchical phrases are involved, several block joinings may take place at once during a single rule application. Figure 1 gives an illustration with an embedding of a lexical phrase (light) in a hierarchical phrase (dark). The gap in the hierarchical phrase $\langle f_1 f_2 X^{\sim 0}, e_1 X^{\sim 0} e_3 \rangle$ is filled with the lexical phrase $\langle f_3, e_2 \rangle$. The discriminative reordering model scores the orientation of the lexical phrase with regard to the neighboring block of the hierarchical phrase which precedes it within the target sequence (here: right orientation), and the block of the hierarchical phrase which succeeds the lexical phrase with regard to the latter (here: left orientation).

5. Soft String-to-Dependency Hierarchical Machine Translation

String-to-dependency hierarchical machine translation (Shen et al., 2008, 2010) employs target-side dependency features to capture syntactically motivated relations between words even across longer distances. It implements enhancements to the hierarchical phrase-based paradigm that allow for an integration of knowledge obtained from dependency parses of the training material. Jane realizes a non-restrictive approach that does not prohibit the production of hypotheses with malformed dependency relations (Stein et al., 2010). Jane includes a spectrum of soft string-to-dependency features: invalidity markers for extracted phrase dependency structures, penalty features for construction errors of the dependency tree assembled during de-

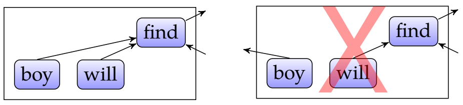


Figure 2. Fixed on head structure (left) and a counterexample (right).

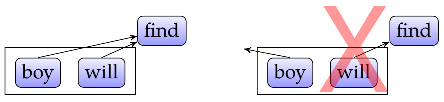


Figure 3. Floating with children structure (left) and a counterexample (right).

coding, and dependency LM features. Dependency trees over translation hypotheses are built on-the-fly during the decoding process from information gathered in the training phase and stored in the phrase table. The soft string-to-dependency features are applied to rate the quality of the constructed tree structures. With version 2 of Jane, dependency LM scoring is—like the other features—directly integrated into the decoder (Peter et al., 2011).

5.1. Dependency Structures in Translation

A dependency models a linguistic relationship between two words, like e.g. the subject of a sentence that depends on the verb. String-to-dependency machine translation demands the creation of dependency structures over hypotheses produced by the decoder. This can be achieved by parsing the training material and carrying the dependency structures over to the translated sentences by augmenting the entries in the phrase table with dependency information. However, the dependency structures seen on phrase level during phrase extraction are not guaranteed to be applicable for the assembling of a dependency tree during decoding. Many of the extracted phrases may be covered by structures where some of the dependencies contradict each other. Dependency structures over extracted phrases which can be considered uncritical in this respect are called *valid*. Valid dependency structures are of two basic types: *fixed on head* or *floating with children*. An example and a counterexample for each type are shown in Figures 2 and 3, respectively. In an approach without hard restrictions, all kinds of structures are allowed, but invalid ones are penalized. Merging heuristics allow for a composition of malformed dependency structures.

A soft approach means that we will not be able to construct a well-formed tree for all translations and that we have to cope with merging errors. During decoding, the previously extracted dependencies are used to build a dependency tree for each hypothesis. While in the optimal case the child phrase merges seamlessly into the parent phrase, often the dependencies will contradict each other and we have to devise strategies for these errors. An example of an ideal case is shown in Figure 4, and a phrase that breaks the previous dependency structure is shown in Figure 5. As a remedy, whenever the direction of a dependency within the child phrase points to the opposite direction of the parent phrase gap, we select the parental direction, but penalize the merging error. In a restrictive approach, the problem can be avoided

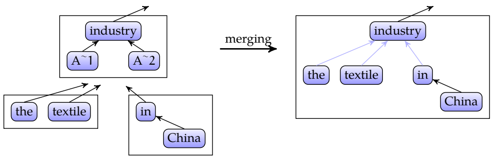


Figure 4. Merging two phrases without merging errors. All dependency pointers point into the same directions as the parent-dependencies.

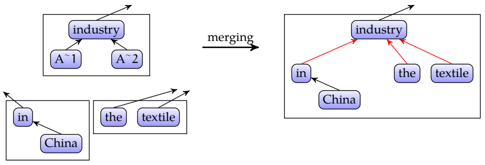


Figure 5. Merging two phrases with one left and two right merging errors. The dependency pointers point into other directions as the parent-dependencies.

by requiring the decoder to always obey the dependency directions of the extracted phrases while assembling the dependency tree.

5.2. Dependency Language Model

Jane computes several language model scores for a given tree: for each node as well as for the left and right-hand side dependencies of each node. For each of these scores, Jane also increments a distinct word count, to be included in the log-linear model, for a total of six features. Note that, while in a well-formed tree only one root can exist, we might end up with a forest rather than a single tree if several branches cannot be connected properly. In this case, the scores are computed on each resulting (partial) tree but treated as if they were computed on a single tree.

6. Experimental Evaluation

We present empirical results obtained with the different models on the Chinese→English 2008 NIST task.

We work with a parallel training corpus of 3.0M Chinese-English sentence pairs (77.5M Chinese / 81.0M English running words). The English target side of the data is lowercased, truecasing is part of the postprocessing pipeline. Word alignments are created by aligning the data in both directions with GIZA++ and symmetrizing the two trained alignments (Och and Ney, 2003). We rely on the Stanford Dependency Parser (Klein and Manning, 2003) to create dependency annotation on the target side of the training data. When extracting phrases, we apply several restrictions, in particular a maximum length of 10 on source and target side for lexical phrases, a length limit of five (including non-terminal symbols) for hierarchical phrases, and no more than two gaps per phrase. The language model is a 4-gram with modified Kneser-Ney smoothing which was trained with the SRILM toolkit (Stolcke, 2002).

We use the cube pruning algorithm (Huang and Chiang, 2007) to carry out the search. A maximum length constraint of 10 is applied to all non-terminals but the initial symbol S. Model weights are optimized against BLEU (Papineni et al., 2002) with

	MT06 (Dev)		MT08 (Test)	
	BLEU [%]	TER [%]	BLEU [%]	TER [%]
Baseline 1 (with s2t+t2s RF word lexicons, $t_{\text{Norm}}(\cdot)$)	32.6	61.2	25.2	66.6
+ s2t+t2s insertion model (RF, individual)	32.9	61.4	25.7	66.2
+ s2t+t2s deletion model (RF, histogram 10)	32.9	61.4	26.0	66.1
+ sentence-level s2t IBM-1, $t_{\text{Norm}}(\cdot)$	32.9	61.6	25.7	66.6
+ phrase-level s2t IBM-1, $t_{\text{Norm}}(\cdot)$	33.0	61.4	26.4	66.1
+ phrase-level t2s IBM-1, $t_{\text{Norm}}(\cdot)$	33.4	60.7	26.5	65.7
+ phrase-level s2t+t2s IBM-1, $t_{\text{Norm}}(\cdot)$	33.8	60.5	26.9	65.4
+ discrim. RO	33.0	61.3	25.8	66.0
+ swap rule + binary swap feature	33.2	61.3	26.2	66.1
+ jump rules + distance-based distortion costs	33.2	61.0	26.4	66.0
+ insertion model + discrim. RO + DWL + triplets	35.0	59.5	27.8	64.4
Soft string-to-dependency	33.5	60.8	26.0	65.7
— only valid phrases	32.8	62.0	25.4	67.1
— no merging errors	32.5	61.5	25.5	66.4
Baseline 2 (no phrase table smoothing)	32.0	62.2	24.3	67.8
+ phrase-level s2t+t2s RF word lexicons, $t_{\text{Norm}}(\cdot)$	32.6	61.2	25.2	66.6
+ phrase-level s2t+t2s RF word lexicons, $t_{\text{NoNorm}}(\cdot)$	32.7	61.8	25.6	66.7
+ phrase-level s2t+t2s RF word lexicons, $t_{\text{NoisyOr}}(\cdot)$	32.4	61.2	25.5	66.4
+ phrase-level s2t+t2s RF word lexicons, $t_{\text{Moses}}(\cdot)$	32.7	61.8	25.4	66.9

Table 1. Experimental results for the NIST Chinese→English translation task (truecase). $s2t$ denotes source-to-target scoring, $t2s$ target-to-source scoring.

MERT on 100-best lists. We employ MT06 as development set, MT08 is used as unseen test set. Translation quality is measured in truecase with BLEU and TER (Snover et al., 2006). The empirical results are presented in Table 1. By incorporating a combination of several of the advanced methods provided by Jane 2 (insertion model, discrim. RO, DWL, triplets), we are able to achieve a performce gain of +2.6% BLEU/ -2.2% TER absolute over a standard hierarchical baseline (*Baseline 1*) .

7. Conclusion

Jane is a stable and efficient state-of-the-art statistical machine translation toolkit that is freely available to the scientific community. It implements the standard hierarchical phrase-based translation approach with many extensions that further enhance the performance of the system. Version 2 of Jane features novel techniques like insertion and deletion models, lexical scoring variants, discriminative reordering extensions, and soft string-to-dependency hierarchical machine translation. We found them to be useful to achieve competitive results on large-scale tasks, and we hope that fellow researchers will benefit from the release of our toolkit.

Acknowledgments

This work was partly achieved as part of the Quaero Programme, funded by OSEO, French State agency for innovation, and partly funded by the European Union under the FP7 project T4ME Net, Contract No. 249119.

Bibliography

Berger, Adam L., Stephen A. Della Pietra, and Vincent J. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–72, Mar. 1996.

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

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

Chen, Stanley F. and Ronald Rosenfeld. A Gaussian Prior for Smoothing Maximum Entropy Models. Technical Report CMUCS-99-108, Carnegie Mellon University, Pittsburgh, PA, USA, Feb. 1999.

Chiang, David. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proc. of the Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 263–270, Ann Arbor, MI, USA, June 2005.

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

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

Darroch, John N. and Douglas Ratcliff. Generalized Iterative Scaling for Log-Linear Models. *Annals of Mathematical Statistics*, 43:1470–1480, 1972.

He, Zhongjun, Yao Meng, and Hao Yu. Maximum Entropy Based Phrase Reordering for Hierarchical Phrase-based Translation. In *Proc. of the Conf. on Empirical Methods for Natural Language Processing (EMNLP)*, pages 555–563, Cambridge, MA, USA, Oct. 2010a.

He, Zhongjun, Yao Meng, and Hao Yu. Extending the Hierarchical Phrase Based Model with Maximum Entropy Based BTG. In *Proc. of the Conf. of the Assoc. for Machine Translation in the Americas (AMTA)*, Denver, CO, USA, Oct./Nov. 2010b.

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

Huck, Matthias and Hermann Ney. Insertion and Deletion Models for Statistical Machine Translation. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Assoc. for Computational Linguistics (HLT-NAACL)*, pages 347–351, Montréal, Canada, June 2012.

Huck, Matthias, Martin Ratajczak, Patrick Lehnen, and Hermann Ney. A Comparison of Various Types of Extended Lexicon Models for Statistical Machine Translation. In *Proc. of the Conf. of the Assoc. for Machine Translation in the Americas (AMTA)*, Denver, CO, USA, Oct./Nov. 2010.

Huck, Matthias, Saab Mansour, Simon Wiesler, and Hermann Ney. Lexicon Models for Hierarchical Phrase-Based Machine Translation. In *Proc. of the Int. Workshop on Spoken Language Translation (IWSLT)*, pages 191–198, San Francisco, CA, USA, Dec. 2011.

Huck, Matthias, Stephan Peitz, Markus Freitag, and Hermann Ney. Discriminative Reordering Extensions for Hierarchical Phrase-Based Machine Translation. In *Proc. of the 16th Annual Conf. of the European Assoc. for Machine Translation*, pages 313–320, Trento, Italy, May 2012.

Klein, Dan and Christopher D. Manning. Accurate Unlexicalized Parsing. In *Proc. of the Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 423–430, Sapporo, Japan, July 2003.

Koehn, Philipp, Franz Joseph Och, and Daniel Marcu. Statistical Phrase-Based Translation. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Assoc. for Computational Linguistics (HLT-NAACL)*, pages 127–133, Edmonton, Canada, May/June 2003.

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

Li, Junhui, Zhaopeng Tu, Guodong Zhou, and Josef van Genabith. Using Syntactic Head Information in Hierarchical Phrase-Based Translation. In *Proc. of the Workshop on Statistical Machine Translation (WMT)*, pages 232–242, Montréal, Canada, June 2012.

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

Nelder, John A. and Roger Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7:308–313, 1965.

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

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

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proc. of the Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, PA, USA, July 2002.

Peter, Jan-Thorsten, Matthias Huck, Hermann Ney, and Daniel Stein. Soft String-to-Dependency Hierarchical Machine Translation. In *International Workshop on Spoken Language Translation*, pages 246–253, San Francisco, CA, USA, Dec. 2011.

Sankaran, Baskaran and Anoop Sarkar. Improved Reordering for Shallow-*n* Grammar based Hierarchical Phrase-based Translation. In *Proc. of the Human Language Technology Conf. /*

North American Chapter of the Assoc. for Computational Linguistics (HLT-NAACL), pages 533–537, Montréal, Canada, June 2012.

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

Shen, Libin, Jinxi Xu, and Ralph Weischedel. String-to-Dependency Statistical Machine Translation. *Computational Linguistics*, 36(4):649–671, Dec. 2010.

Snover, Matthew, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A Study of Translation Edit Rate with Targeted Human Annotation. In *Proc. of the Conf. of the Assoc. for Machine Translation in the Americas (AMTA)*, pages 223–231, Cambridge, MA, USA, Aug. 2006.

Stein, Daniel, Stephan Peitz, David Vilar, and Hermann Ney. A Cocktail of Deep Syntactic Features for Hierarchical Machine Translation. In *Proc. of the Conf. of the Assoc. for Machine Translation in the Americas (AMTA)*, Denver, CO, USA, Oct./Nov. 2010.

Stein, Daniel, David Vilar, Stephan Peitz, Markus Freitag, Matthias Huck, and Hermann Ney. A Guide to Jane, an Open Source Hierarchical Translation Toolkit. *The Prague Bulletin of Mathematical Linguistics*, (95):5–18, Apr. 2011.

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

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

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

Vilar, David and Hermann Ney. Cardinality pruning and language model heuristics for hierarchical phrase-based translation. *Machine Translation*, Nov. 2011. URL `http://dx.doi.org/10.1007/s10590-011-9119-4`.

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

Vilar, David, Daniel Stein, Matthias Huck, and Hermann Ney. Jane: Open Source Hierarchical Translation, Extended with Reordering and Lexicon Models. In *Proc. of the Workshop on Statistical Machine Translation (WMT)*, pages 262–270, Uppsala, Sweden, July 2010a.

Vilar, David, Daniel Stein, Stephan Peitz, and Hermann Ney. If I Only Had a Parser: Poor Man's Syntax for Hierarchical Machine Translation. In *Proc. of the Int. Workshop on Spoken Language Translation (IWSLT)*, pages 345–352, Paris, France, Dec. 2010b.

Vilar, David, Daniel Stein, Matthias Huck, and Hermann Ney. Jane: an advanced freely available hierarchical machine translation toolkit. *Machine Translation*, 2012. URL `http://dx.doi.org/10.1007/s10590-011-9120-y`.

Zens, Richard and Hermann Ney. Improvements in Phrase-Based Statistical Machine Translation. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Assoc. for Computational Linguistics (HLT-NAACL)*, pages 257–264, Boston, MA, USA, May 2004.

Zens, Richard and Hermann Ney. Discriminative Reordering Models for Statistical Machine Translation. In *Proc. of the Human Language Technology Conf. / North American Chapter of the Assoc. for Computational Linguistics (HLT-NAACL)*, pages 55–63, New York City, NY, USA, June 2006.

Address for correspondence:
Matthias Huck
huck@cs.rwth-aachen.de
Human Language Technology and Pattern Recognition Group
Computer Science Department
RWTH Aachen University
D-52056 Aachen, Germany

The Prague Bulletin of Mathematical Linguistics
NUMBER 98 OCTOBER 2012 51–61

pycdec: A Python Interface to cdec

Victor Chahuneau, Noah A. Smith, Chris Dyer

Language Technologies Institute, Carnegie Mellon University

Abstract

This paper describes `pycdec`, a Python module for the `cdec` decoder. It enables Python code to use `cdec`'s fast C++ implementation of core finite-state and context-free inference algorithms for decoding and alignment. The high-level interface allows developers to build integrated MT applications that take advantage of the rich Python ecosystem without sacrificing computational performance. We give examples of how to interact directly with the main `cdec` data structures (lattices, hypergraphs, sparse feature vectors), evaluate translation quality, and use the suffix-array grammar extraction code. This permits rapid prototyping of new algorithms for training, data visualization, and utilizing MT and related structured prediction tasks.

1. Introduction

Machine translation decoders are complex pieces of software. They must provide efficient search and inference algorithms, represent large translation grammars (e.g., phrase tables), and support scoring of hypotheses with a variety of feature functions. Typically, they also contain functionality for parameter learning and translation quality evaluation. Despite this sophistication, machine translation can be formalized quite well using familiar, well-defined mathematical objects (e.g., lattices, vectors, hypergraphs, weighted finite-state transducers) and in terms of just a few algorithms (e.g., FST/CFG intersection, shortest path search, etc.).

Although this convenient and precise mathematical language exists (and is, of course, used in the academic literature), the programmatic interfaces to real translation systems are much more complicated. On one hand, the low-level implementation in the decoder's native language (usually C++ or Java) is highly optimized, making the mapping between the mathematical primitives discussed in papers and the actual code difficult to perceive. On the other hand, the high-level command-line interface

Corresponding author: vchahune@cs.cmu.edu
Cite as: Victor Chahuneau, Noah A. Smith, Chris Dyer. pycdec: A Python Interface to cdec. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 51–61. doi: 10.2478/v10108-012-0008-7.

that decoders expose is not suitably expressive for anything but the most coarse automation. As a result, when new researchers and engineers master the theory of MT, they must still invest a great deal of work in learning a real software system before they can really innovate. This paper describes a new Python interface for the `cdec` decoder designed to narrow the gap between theory and practice.

`cdec` is a good candidate for this task because it has been designed with modularity in mind from the beginning (Dyer et al., 2010). We choose Python as the language to expose the API for its large user base and rich extension ecosystem, and also because it is an interpreted language supporting both object-oriented and functional programming. The goals for this project include:

- exposing the decoder functionality as a library with a natural, easy-to-understand interface;
- providing access to the decoder's data structures, including translation hypergraphs, input lattices, hypothesis feature vectors, etc.;
- allowing direct integration of external Python libraries such as *NLTK* (Bird et al., 2009) and *scikit-learn* (Pedregosa et al., 2011) into machine translation systems; and
- encouraging creative use of machine translation technology by programmers who do not need to learn the details of open-source machine translation systems.

The `pycdec` interface is implemented using Cython (Behnel et al., 2011) and included as part of the open-source `cdec` distribution.[1] In the following, we give an introduction to its main functionality and then describe a few applications of the new interface.

2. Related Work

Experiment management tools (Koehn, 2010; Clark et al., 2010) abstract the internals of the decoder from the user to provide a uniform interface to the main training steps of the system. While these facilitate the coordination of large experimental setups, they must be configured using either a domain-specific language or a graphical interface that the user has to learn to manipulate the system. We go in the opposite direction and directly expose the decoder to the user in a modern and familiar language, Python.

Recent work has also explored the use of visualization tools for machine translation. Weese and Callison-Burch (2010) describe extensions to the Joshua decoder to populate a graphical interface used to display derivation trees and hypergraphs. We obtain similar functionality with just a couple of lines of `pycdec` in conjunction with existing visualization tools (§ 4.1). Since our visualizations are computed with simple Python scripts, developers have far more flexibility to innovate.

Finally, the popularity of web translation services such as Google Translate has motivated the development of web interfaces for open-source translation tools (Fed-

[1] `http://cdec-decoder.org`

ermann and Eisele, 2010). We demonstrate how such tools can be rapidly developed using common networking and communication libraries (§ 4.2).

3. Library Description

The API of pycdec exposes the main data structures and algorithms necessary for machine translation and similar structured prediction problems. When it makes sense to do so, we retain the structure of the C++ interface, but otherwise follow the Python conventions.

3.1. Basic Translation and Inference API

The translation interface is provided by the Decoder class. The constructor takes arguments specifying the configuration of the decoder. Feature weights used by the decoder can be assigned and modified at any time (for example, in an online training algorithm).

Once the decoder is instantiated, it can translate sentences, optionally using a sentence-specific grammar passed as a string argument. The result returned is a translation hypergraph encoding the search space explored by the decoder.

The Hypergraph object is central to this system, and therefore it supports several types of operations:

- extraction of the Viterbi translation (viterbi), source and target trees (viterbi_trees) and of the corresponding feature vector (viterbi_features);
- extraction of k-best translations (kbest), source and target trees (kbest_trees) and of the corresponding feature vectors (kbest_features);
- operations that modify the hypergraph, including:
  - rescoring with new weights (reweight),
  - inside-outside pruning (prune),
  - intersection with a reference sentence or lattice (intersect); and
- iteration over the edges and nodes that form the hypergraph.

As an example, here is how to use a hierarchical phrase-based decoder to translate a sentence with a grammar read from a file:

```python
import cdec
# Create and configure a decoder object
decoder = cdec.Decoder(formalism='scfg',
        feature_function=['WordPenalty', 'KLanguageModel lm.klm'],
        add_pass_through_rules=True)
# Set weights for the language model features
decoder.weights['LanguageModel_OOV'] = -1
decoder.weights['LanguageModel'] = 0.1
# Read a SCFG from a file
grammar = open('grammar.scfg').read()
```

```python
# Translate the sentence; returns a translation hypergraph
hg = decoder.translate('traduttore , traditore .', grammar=grammar)
# Extract the best hypothesis from the hypergraph
print(hg.viterbi())
```

Other formalisms such as phrase-based translation can be accessed in a similar way by setting the appropriate configuration parameters for the decoder.

3.2. Grammar Extraction API

To minimize memory usage and code complexity, cdec uses *per-sentence grammars* (i.e., grammars containing just the rules that can match the words in a single test sentence). While these grammars can be constructed from arbitrary tools, pycdec includes the suffix array grammar extractor of Lopez (2007), which uses an efficient compiled representation of a parallel corpus and word alignment to construct translation grammars on demand for new test sentences. The Python module makes this online grammar extraction procedure particularly simple.

After the training corpus has been compiled into a suffix array representation using the tools distributed with cdec, the resulting configuration can be used to call the grammar extractor for any arbitrary input:

```python
extractor = cdec.sa.GrammarExtractor('extractor_config.py')
decoder = cdec.Decoder(formalism='scfg')
sentence = 'traduttore , traditore .'
decoder.translate(sentence, grammar=extractor.grammar(sentence))
```

The extraction algorithm is implemented in Cython and is suitable for online extraction of grammars from very large corpora (Lopez, 2008).

3.3. Translation Quality Evaluation

cdec includes implementations of basic evaluation metrics (BLEU and TER), exposed in Python via the cdec.score module. For a given (reference, hypothesis) pair, sufficient statistics vectors (SufficientStats) can be computed. These vectors are then added together for all sentences in the corpus and the final result is finally converted into a real-valued score.

Writing a script which computes the BLEU score for a set of hypotheses and references is thus straightforward:

```python
import cdec.score
with open('hyp.txt') as hyp, open('ref.txt') as ref:
    stats = sum(cdec.score.BLEU(r).evaluate(h) for h, r in zip(hyp, ref))
    print('BLEU = {0:.1f}'.format(stats.score * 100))
```

Multiple references can be used by supplying a list of strings instead of a single string:

```python
cdec.score.BLEU([r1, r2])
```

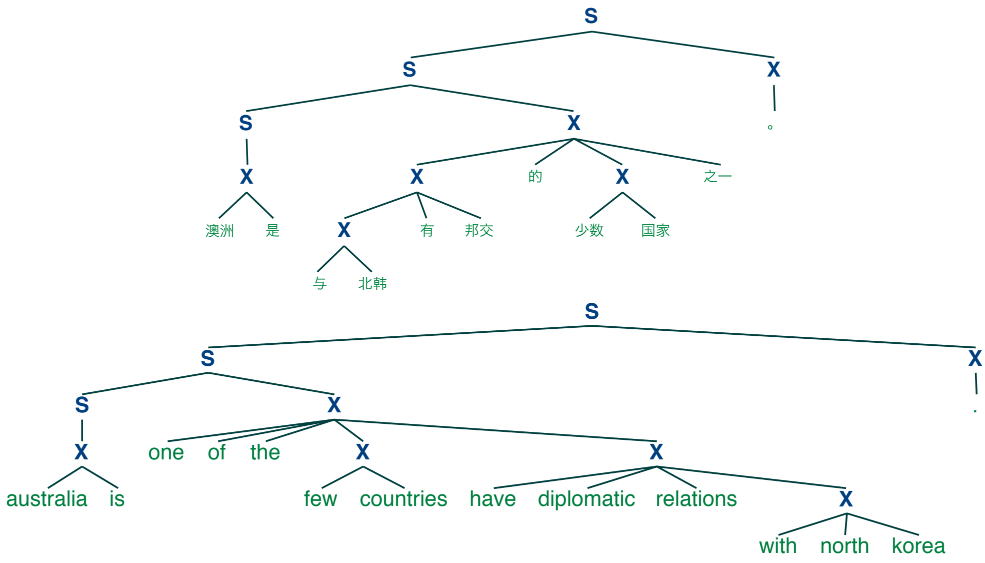


Figure 1. Source (Chinese) and target (English) parse trees, drawn using NLTK

When implementing training algorithms using `pycdec` (§ 4.3), it is often necessary to manipulate k-best lists of scored hypotheses. For every metric, sentence scorers are able produce such sets of hypotheses (`CandidateSet`). For each `Candidate` in the list, its sentence-level metric score (`score`), feature vector (`fmap`) and output string (`words`) can be obtained.

4. Applications

In this section, we provide several examples using the `pycdec` module to solve visualization, parameter estimation, and grammar extraction problems.

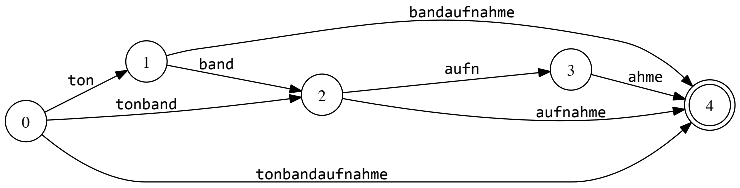
4.1. Visualizing the Result of Decoding

We can make use of the functionality of *NLTK* to visualize derivation trees that result from the decoding of a sentence under a synchronous grammar. Fig. 1 shows an example for a Chinese/English hierarchical phrase-based system. The corresponding Python code is:

```
hg = decoder.translate(sentence)
f_tree, e_tree = hg.viterbi_trees()
nltk.Tree(f_tree).draw() # draw source tree
nltk.Tree(e_tree).draw() # draw target tree
```

Another finite-state formalism supported by `cdec` is compound splitting, in which case the model output takes the form of a lattice (encoded as a hypergraph produced by the `translate` method). Conversion to the Graphviz dot format (Ellson et al., 2003) allows a compact visualization of the output space. Then we can use any of the several Python interfaces to Graphviz to directly render the lattice as shown below:

```
hg = decoder.translate('tonbandaufnahme')
hg.prune(beam_alpha=9.0, csplit_preserve_full_word=True)
pydot.graph_from_dot_data(hg.lattice().todot()).write_svg('lattice.svg')
```



Finally, we introduce a more complex visualization which makes use of the direct access to the hypergraph (Fig. 2). For the same sentence as our first example, we represent the synchronous parse chart as a table, with each cell containing all the possible non-terminals for the corresponding span. Then we color the background of the cell according to the following value:

$$\log \sum_{node \in nodes} \max_{edge \to node} p(edge)$$

This gives an indication of how much uncertainty is present at each level of the parse. We believe that this is an efficient method to compactly visualize the enormous output space produced by the decoder: the hypergraph contains 244, 232 edges and 77 nodes encoding a total of 3.8×10^{28} paths!

We conclude by noting that, as opposed to specialized visualization tools (e.g., Weese and Callison-Burch, 2010), `pycdec` allows the programmer to use any algorithm and output format to explore the various decoder data structures. We suggest in particular the use of the IPython notebook (Pérez and Granger, 2007) to produce HTML or SVG graphics directly in a web browser, as we did for Fig. 2.

4.2. A Web Translation Interface

Commercial web translation platforms, such as Google Translate, have been very successful in bringing state of the art machine translation systems to internet users. In a research environment, it can also be useful to provide similar web interfaces, for example, for non-technical users to explore the strength and weaknesses of the system.

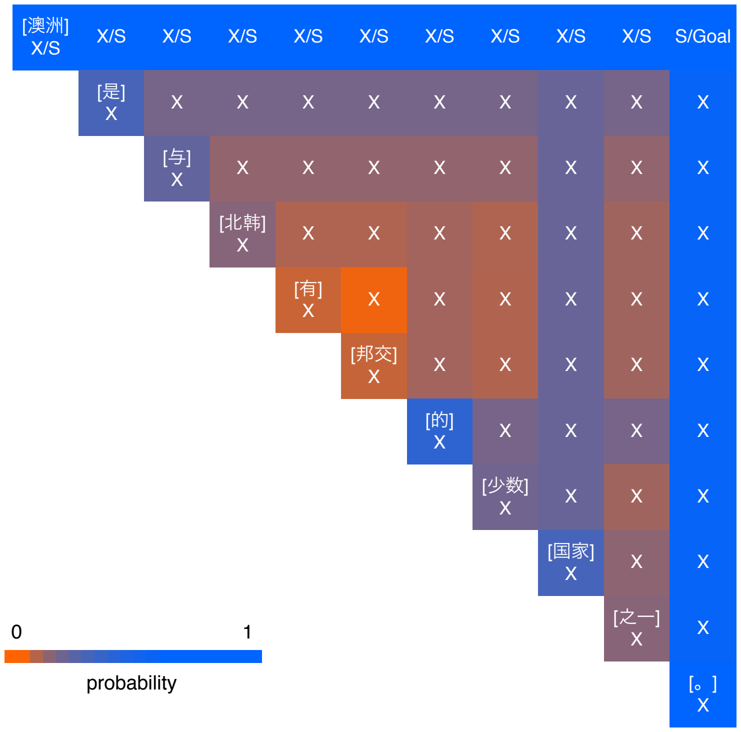


Figure 2. Chart for the synchronous parse of a Chinese sentence

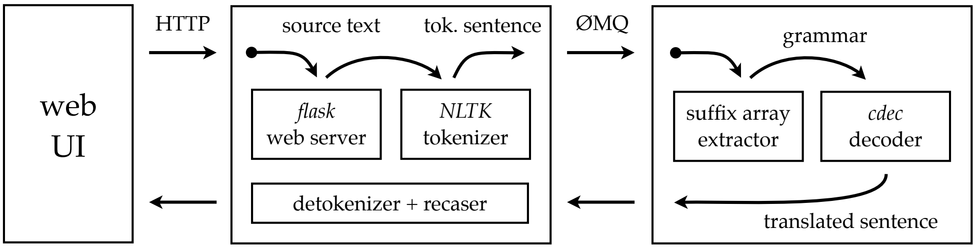


Figure 3. Architecture of the web translation service

Since pycdec provides an access to the decoder directly from Python, it is possible to implement such a service with standard networking libraries to manage communication. Fig. 3 illustrates the messages transmitted between the three layers of the architecture as a piece of text is translated:

- The user interface consists of a HTML page with a JavaScript UI interacting with the web server via asynchronous HTTP requests.
- When the web server – a Python application implemented using the *flask* web framework – receives a translation request, it applies standard pre-processing steps to the input. The text is first segmented into sentences, and each sentence is in turn tokenized. We rely on *NLTK* for this step, at least when the source language is English. Then, each sentence is sent separately through a ZeroMQ[2] socket to the translation server, using the *pyzmq* library.
- The translation server receives individual sentences, for which it extracts grammars on the fly as explained in § 3.2, before calling the decoder to translate the sentence with the extracted grammar. It replies to the web server with the translated sentence.
- The web server then post-processes each translated sentence and recomposes the translated text block before transmitting it back to the web UI.

Even with such a minimal architecture, our system can easily be scaled by multiplying the number of translation servers and relying on ZeroMQ to distribute translation tasks to the multiple decoder instances.

4.3. Parameter Estimation

Another natural use case for pycdec is to facilitate development of new discriminative parameter learning algorithms in Python. Such algorithms (e.g., Chiang et al., 2008; Hopkins and May, 2011; Gimpel and Smith, 2012) use the decoder to compute statistics over the hypergraphs or k-best lists produced by decoding a development set so as to optimize some objective function (like BLEU, or likelihood). In these algorithms, the majority of the computational effort is the decoding step (or a similar inference problem, such as computing posterior probabilities over n-grams), whereas the manipulation of the weight vector is inexpensive. Thus, a natural division of labor is to use Python's mathematical libraries for manipulation of the weight vector and pycdec for inference.

Advantages of writing a new training method with pycdec include the possibility to easily debug code by directly interacting with the decoder data structures through the Python interpreter, and the availability of mature machine learning libraries such as *scikit-learn*.

To illustrate these claims, we implement a recently published training method that is not currently included in cdec. We choose Bazrafshan et al. (2012), a simple exten-

[2]http://www.zeromq.org

```
decoder = cdec.Decoder(...)

def get_pairs(source, reference):
    hg = decoder.translate(source)
    # 1. Generate a list containing the k best translations
    cs = cdec.score.BLEU(reference).candidate_set()
    cs.add_kbest(hg, K)
    # 2. Use the uniform distribution to sample n random pairs
    # from the set of candidate translations
    pairs = []
    for _ in range(n_samples):
        ci = cs[random.randint(0, len(cs) - 1)]
        cj = cs[random.randint(0, len(cs) - 1)]
        # 3. Keep a pair of candidates if the difference between their score
        # is bigger than a threshold t
        if abs(ci.score - cj.score) < score_threshold: continue
        pairs.append((ci.fmap - cj.fmap, ci.score - cj.score))
    # 4. From the potential pairs kept in the previous step,
    # keep the s pairs that have the highest score
    for x, y in heapq.nlargest(n_pairs, pairs, key=lambda xy: abs(xy[1])):
        # 5. For each pair kept in step 4, make two data points
        yield x, y
        yield -1 * x, -1 * y

# The DictVectorizer converts dictionaries into sparse vectors
vectorizer = sklearn.feature_extraction.DictVectorizer()

for _ in range(n_iterations):
    # Collect training pairs
    X, g = [], []
    for source, reference in zip(sources, references):
        for x, y in get_pairs(source, reference):
            X.append(dict(x))
            g.append(y)
    # Train a linear regression model
    model = sklearn.linear_model.LinearRegression()
    X = vectorizer.fit_transform(X)
    model.fit(X, g)
    # Update weights with the learned model
    for fname, fval in zip(vectorizer.feature_names_, model.coef_):
        decoder.weights[fname] = (alpha * fval +
                (1 - alpha) * decoder.weights[fname])
```

Figure 4. Python code for Tuning as Linear Regression (Bazrafshan et al., 2012)

sion to PRO (Hopkins and May, 2011) which uses linear regression instead of a binary classifier to rank sampled training pairs (briefly: the model is trained to predict the difference in sentence level BLEU scores based on a difference in feature vectors). The complete Python code is given in Fig. 4.

5. Conclusion

We have presented `pycdec`, a high-level Python interface to the fast `cdec` decoder. We illustrated how such an interface allows effortless development of visualizations, training algorithms and applications using machine translation. We hope that the release of our tool will encourage further creative uses of finite-state and context-free methods for machine translation and related applications.

Acknowledgments

This research is supported by the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-10-1-0533.

Bibliography

Bazrafshan, M., T. Chung, and D. Gildea. Tuning as linear regression. In *Proc. of NAACL-HLT*, pages 543–547. Association for Computational Linguistics, 2012.

Behnel, S., R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31–39, March–April 2011.

Bird, S., E. Klein, and E. Loper. *Natural language processing with Python*. O'Reilly Media, 2009. URL `http://nltk.org`.

Chiang, D., Y. Marton, and P. Resnik. Online large-margin training of syntactic and structural translation features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 224–233. Association for Computational Linguistics, 2008.

Clark, J.H., J. Weese, B.G. Ahn, A. Zollmann, Q. Gao, K. Heafield, and A. Lavie. The machine translation toolpack for LoonyBin: Automated management of experimental machine translation hyperworkflows. *The Prague Bulletin of Mathematical Linguistics*, 93:117–126, 2010.

Dyer, C., J. Weese, H. Setiawan, A. Lopez, F. Ture, V. Eidelman, J. Ganitkevitch, P. Blunsom, and P. Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proc. of the ACL (Demonstration track)*, pages 7–12. Association for Computational Linguistics, 2010.

Ellson, J., E.R. Gansner, E. Koutsofios, S.C. North, and G. Woodhull. Graphviz and Dynagraph – static and dynamic graph drawing tools. In Junger, M. and P. Mutzel, editors, *Graph Drawing Software*, pages 127–148. Springer-Verlag, 2003. URL `http://graphviz.org`.

Federmann, C. and A. Eisele. MT server land: An open-source MT architecture. *The Prague Bulletin of Mathematical Linguistics*, 94:57–66, 2010.

Gimpel, K. and N.A. Smith. Structured ramp loss minimization for machine translation. In *Proceedings of NAACL*, 2012.

Hopkins, M. and J. May. Tuning as ranking. In *Proc. of EMNLP*, pages 1352–1362. Association for Computational Linguistics, 2011.

Koehn, P. An experimental management system. *The Prague Bulletin of Mathematical Linguistics*, 94:87—96, 2010.

Lopez, A. Hierarchical phrase-based translation with suffix arrays. In *Proc. of EMNLP-CoNLL*, pages 976–985, 2007.

Lopez, A. Tera-scale translation models via pattern matching. In *Proc. COLING*, pages 505–512, 2008.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL `http://scikit-learn.org`.

Pérez, F. and B.E. Granger. IPython: a system for interactive scientific computing. *Comput. Sci. Eng.*, 9(3):21–29, 2007. URL `http://ipython.org`.

Weese, J. and C. Callison-Burch. Visualizing data structures in parsing-based machine translation. *The Prague Bulletin of Mathematical Linguistics*, 93:127–136, 2010.

Address for correspondence:
Victor Chahuneau
`vchahune@cs.cmu.edu`
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 63–74

Phrasal Rank-Encoding: Exploiting Phrase Redundancy and Translational Relations for Phrase Table Compression

Marcin Junczys-Dowmunt

Information Systems Laboratory, Adam Mickiewicz University, Poznań, Poland
Global Databases Service, World Intellectual Property Organization (WIPO), Geneva, Switzerland

Abstract

We describe Phrasal Rank-Encoding (PR-Enc), a novel method for the compression of word-aligned target language data in phrase tables as used in phrase-based SMT. This method reduces the redundancy in phrase tables which is a direct effect of the phrase-based approach. A combination of PR-Enc with Huffman coding allows to reduce the size of an aggressively compressed phrase table by another 39 percent. Using this and other methods for space reduction in a new binary phrase table implementation, a size reduction by an order of magnitude is achieved when comparing to the Moses on-disk phrase table implementation. Concerning decoding speed, all variants of the new phrase table are faster than the Moses binary phrase table implementation while the PR-Enc encoded variant outperforms all other methods.

1. Introduction

Phrase tables as used in phrase-based statistical machine translation (PB-SMT) are huge. Their size is a direct consequence of the PB-SMT approach itself and the fact that precomputed phrase pairs can be accessed efficiently. Precomputation, however, leads to a combinatorial bloat of phrases and to phrase redundancy since for any phrase pair all possible subphrase pairs may be included in the phrase table.

Explicitly stored phrase tables are currently the most widely used representation of translation models in PB-SMT. The main goal of this paper is to describe Phrasal Rank-Encoding (PR-Enc) — a lossless encoding and compression method dedicated to explicitly stored phrase tables. PR-Enc aims to reduce redundancy by exploiting the phrase table itself as a compression dictionary, making use of translational relations

Corresponding author: junczys@amu.edu.pl
Cite as: Marcin Junczys-Dowmunt. Phrasal Rank-Encoding: Exploiting Phrase Redundancy and Translational Relations for Phrase Table Compression. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 63–74. doi: 10.2478/v10108-012-0009-6.

between subphrases and the repetitiveness of subphrases. Previous approaches concentrate only on one of these properties and correspondingly either require additional resources, like external translation dictionaries, or is restricted to a certain type of phrasal redundancy, for instance prefixes in trie-based implementations. An efficient dynamic programming algorithm for the decompression of PR-Enc encoded target phrases is presented. The described method is employed in a previously introduced compact phrase table implementation (Junczys-Dowmunt, 2012a,b) for Moses (Koehn et al., 2007) that can be used as a replacement for the current binary phrase table.

2. Related Work

2.1. Previous Work in Machine Translation

Zens and Ney (2007) describe a phrase table architecture on which the standard binary phrase table of Moses is based. Memory requirements are low due to on-demand loading, disk space requirements, however, can become substantial. The only compression technique relies on the application of a trie for the representation of a source phrase index which collapses common source phrase prefixes into single paths.

Germann et al. (2009) introduce tightly packed tries (TBT) which they use for language models and phrase tables in the Portage SMT system (Sadat et al., 2005). TBTs are stored in byte arrays with variable byte encoding applied to reduce their space requirements. The section on phrase tables does not provide sufficient information to compare our approaches, but we have contacted the authors and may be able to provide a comparison of compression rates in the future.

Suffix-array based implementations of translation models have been introduced by Callison-Burch et al. (2005). Both sides of a parallel corpus are stored as suffix arrays. If alignment data between the parallel sentences is provided, phrase pairs can be extracted and scored on demand. The precomputation of phrase pairs is avoided altogether, which immediately solves the problem of redundancy. On-demand translation models are a promising alternative to phrase tables, but we do not compare this approach with ours, instead we concentrate on explicitly stored phrase tables.

Phrase table filtering (e.g. Johnson et al., 2007) can be seen as a type of lossy compression. Reduction rates of more than 80 percent while maintaining or even improving translation quality are not uncommon. We find this particularly interesting since a combination of phrase table filtering with our approach can yield a translation model size reduction by two orders of magnitude.

2.2. Compression of Parallel Corpora

Conley and Klein (2008) propose an encoding scheme of target language data based on word alignment and translational relations. However, they require the existence of lemmatizers and a translation lexicon. From the aligned parallel data a lexicon of lemmatized parallel phrases is created. Target phrases are replaced with pointers consisting of start and end indexes of the corresponding source phrase, indexes of translations, and one integer pointer per target word to its inflected form. This turns the method into a word-based method since no length reduction of the text is achieved. Actually, there are more pointers after encoding than there were target words before. Compression is achieved by the application of a Huffman coder.

The most recent work is Sanchez-Martinez et al. (2012) who propose to use "biwords" to compress parallel data sequentially. Similar as in Conley and Klein (2008), translational relations and Huffman coding are employed to take advantage of the improved entropy properties of the encoded data. This method is called Translational Relation Encoding (TRE). Again, mainly word-based translational relations are used, allowing at most many-to-one alignments. Sanchez-Martinez et al. include a list of biwords, a translation dictionary extracted from the alignment, in the compressed file.

2.3. Compact Phrase Table Implementation

Junczys-Dowmunt (2012a) introduces a compact phrase table architecture which we use for our experiments with PR-Enc. A "baseline" variant is presented that uses standard compression methods like the Simple-9 algorithm (Anh and Moffat, 2004), variable-byte encoding (Scholer et al., 2002), and Huffman coding (Huffman, 1952) of target words, scores, and alignment points. Size reduction for source phrases is achieved by using a minimal perfect hash function as an index. Junczys-Dowmunt (2012b) describes the further reduction of the source phrase index and the impact of false positive assignments of target phrases to source phrases on translation quality. That implementation achieves a size reduction of more than 77 percent when compared to the Moses binary phrase table with significantly better performance.

Also in Junczys-Dowmunt (2012a) word-based Rank-Encoding is described. This method is similar to TRE (Sanchez-Martinez et al., 2012), but does not store source words which are provided during phrase table querying. Target phrase words are replaced with pairs of pointers. The first pointer indicates the corresponding source phrase word, the second the rank of the target word among the translations of the source word. A translation lexicon generated from the alignment is included in the phrase table. Again, Huffman coding improves the compression rate.

3. Phrasal Rank Encoding

The general idea of Phrasal Rank-Encoding is similar to that of classic dictionary-based compression methods like LZ77 (Ziv and Lempel, 1977). Repetitive subphrases are replaced by pointers to subphrases in a phrase dictionary which should result in a reduction of data length. Decompression relies on the look-up and reinsertion of subphrases based on the pointer symbols. Something similar, though in an rather ineffective way, has been attempted by Conley and Klein (2008). If we simplify their method by dropping all external data requirements and move it onto the ground of phrase tables, we get a basic version of Phrasal Rank-Encoding. Instead of compressing a bitext with a translation lexicon of phrases, we compress the lexicon itself. In Phrasal Rank-Encoding the compressed phrase table is its own phrase dictionary.

Although Phrasal Rank-Encoding shares some properties with word-based Rank-Encoding and mentioned bitext compression methods, compression is achieved in a different way: sequential data is implicitly turned into a graph-like structure similar to a trie or finite-state automaton, which is more visible during the discussion of the decoding algorithm. Translational relations and entropy coding help to compress the graph structure itself and not so much the data in it, which is not unlikely to the work presented by Germann et al. (2009). Phrasal Rank-Encoding could also be used to turn the target data of the Moses binary phrase table based on Zens and Ney (2007) into a graph-like structure without changing the underlying implementation (much).

3.1. Encoding Procedure

The encoding procedure is presented in Figure 1. PR-Enc requires the phrase table to contain word alignment information. In order to perform encoding efficiently, it should be possible to look-up phrase pairs and to retrieve the rank of a target phrase relative to its corresponding source phrase. By rank we mean the position of a target phrase among a list of phrase pairs with the same source phrase. The list is ordered decreasingly by the translation probability $P(\mathbf{t}|\mathbf{s})$, i.e. the most probable translation has rank 0. In our implementation this is achieved by creating a minimal perfect hash function with concatenations of source and target phrases as keys which are mapped to ranks. This searchable phrase table is passed to the algorithm as RankedPT.

We illustrate the algorithm with an example. Given are a Spanish-English phrase pair and the internal word alignment depicted by the black boxes in Figure 2:

```
es: Maria no daba una bofetada a la bruja verde
en: Mary did not slap the green witch
```

Phrase pairs are represented as quadruples where the values correspond to the source phrase start position, the target phrase start position, the length of the source phrase, and the length of the target phrase. In line 3 of the algorithm, all true subphrase pairs of the encoded phrase pair are computed. The result is marked in Figure 2 by filled and empty rectangles — with one exception: the complete phrase pair itself is ruled out for not being a true subphrase pair. The first condition of the expression in line 3 requires subphrase pairs to lie within the boundaries of the encoded phrase. The second, introduced by Zens et al. (2002), defines subphrase pairs that are consistent with the underlying alignment. The same procedure is used during phrase pair extraction when the translation model is created. In order to avoid self-references, the third condition forbids to add the input phrase pair itself.

Next, the subphrase pairs are inserted into a queue (line 4) according to the following order: subphrase pairs are ordered decreasingly by length and increasingly by

```
Function EncodeTargetPhrase($\mathbf{s}, \mathbf{t}, A, \mathrm{Order}, \mathrm{RankedPT}$)
    $\hat{\mathbf{t}} \leftarrow \langle\,\rangle;\ \hat{A} \leftarrow A$
    $P \leftarrow \{\langle i, j, m, n\rangle : (0 \le i < i+m \le |\mathbf{s}| \wedge 0 \le j < j+n \le |\mathbf{t}|)$
                    $\wedge\ \forall \langle i', j'\rangle \in A : (i \le i' < i+m \Leftrightarrow j \le j' < j+n)$
                    $\wedge\ (m < |\mathbf{s}| \vee n < |\mathbf{t}|)\}$
    $\mathbf{Q} \leftarrow \mathrm{Queue}(P, \mathrm{Order})$
    while $|\mathbf{Q}| > 0$ do
        $\langle i, j, m, n\rangle \leftarrow \mathrm{Pop}(\mathbf{Q})$
        $\mathbf{s}' \leftarrow \mathrm{Substring}(\mathbf{s}, i, m)$
        $\mathbf{t}' \leftarrow \mathrm{Substring}(\mathbf{t}, j, n)$
        if $\exists r : \langle \mathbf{s}', \mathbf{t}', r\rangle \in \mathrm{RankedPT}$ then
            $T[j] \leftarrow \langle i - j, |\mathbf{s}| - (i+m), r\rangle$
            $S[j] \leftarrow n$
            $\hat{A} \leftarrow \hat{A} \setminus \{\langle i', j'\rangle \in \hat{A} : i \le i' < i+m \wedge j \le j' < j+n\}$
            $P \leftarrow P \setminus \{\langle i', j', m', n'\rangle : i \le i' < i+m \vee j \le j' < j+n$
                                    $\vee\ i' \le i < i'+m' \vee j' \le j < j'+n'\}$
            $\mathbf{Q} \leftarrow \mathrm{Queue}(P, \mathrm{Order})$
    $j \leftarrow 0$
    while $j < |\mathbf{t}|$ do
        if $S[j] > 0$ then
            $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{t}} \cdot \langle T[j]\rangle$
            $j \leftarrow j + S[j]$
        else
            $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{t}} \cdot \langle t_j\rangle$
            $j \leftarrow j + 1$
    return $\langle \hat{\mathbf{t}}, \hat{A}\rangle$
```

Figure 1. Algorithm for Phrasal Rank-Encoding

the start position of the target phrase, then by length and start position of the source phrase. For our example, the first phrase pair popped from the queue is

```
es: no daba una bofetada a la bruja verde
en: did not slap the green witch
```

which is checked for inclusion in the ranked phrase table (line 9). A rank of 0 is assigned. The target subphrase is replaced with a pointer symbol

```
es: Maria no daba una bofetada a la bruja verde
en: Mary (0,0,0)
```

The integer values of the pointer triple have the following interpretation:

- The first is the difference of source and target start positions of the subphrase pair. Due to general monotonicity this yields smaller integers than positions.

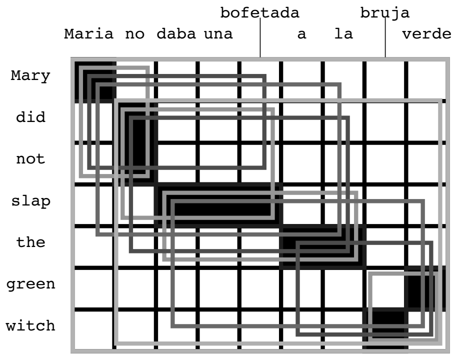


Figure 2. Archetypical example for phrase pair extraction by Knight and Koehn (2003)

- The second value is the distance of the right source subphrase boundary from the end of the encoded phrase.
- The last value is the rank of the selected subphrase pair.

All alignment points lying within the boundaries of the chosen subphrase pair are removed (line 12) and all subphrase pairs that overlap with the subphrase pair are deleted from the queue (line 13 and 14). Only one phrase pair is left in the queue:

```
es: Maria
en: Mary
```

Applying the same procedure again, the following encoded phrase pair is produced:

```
es: Maria no daba una bofetada a la bruja verde
en: (0,8,0) (0,0,0)
```

Target subphrases for which no substitution has been found are kept as plain words.

3.2. Decoding Procedure

A naive decoding procedure processes a mainly-binary tree (Figure 3) in potentially exponential time. However, if all target phrases for a sentence are considered, a dynamic programming algorithm with linear time-complexity per phrase can be constructed. Moses queries the phrase table processing sentences in a left-to-right fashion, starting with subphrases of length 1 and increasing the length until a limit is reached. Then it moves to the next word, starting at length 1. Hence, if a phrase is retrieved, its prefixes have already been processed. If all queried phrases are cached for decoding and all phrases used for decoding are cached for look-up, the total number of phrase table accesses is the same as in a linear phrase table. With caching, a target phrase for “Maria no daba una bofetada” would be found immediately, avoiding the descent into the left branch of the graph. The subphrase “a la” will still be processed, but when Moses queries that phrase, it will be retrieved from the cache.

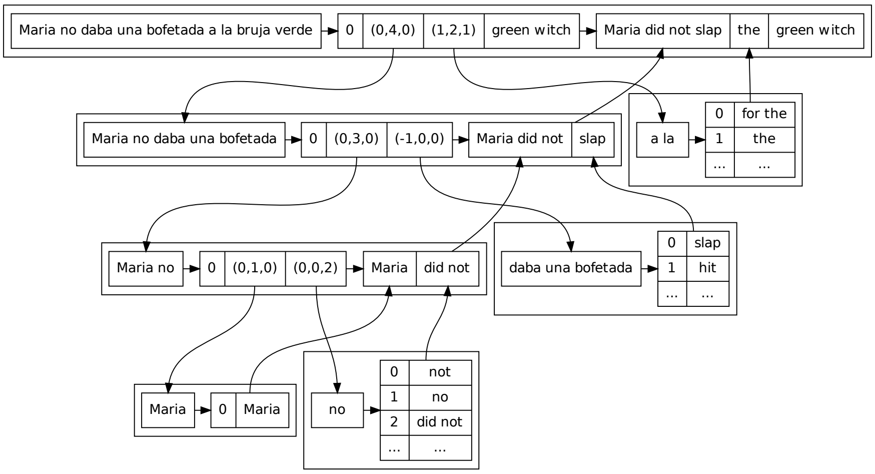


Figure 3. Phrasal Rank-Decoding without Caching

This is implemented in the algorithm in Figure 4. A target phrase collection for a source phrase is created. If a target phrase with the given rank has been seen before, it is retrieved from the cache. Otherwise an encoded version is loaded from the phrase table and passed to DecodeTargetPhrase. If $\hat{t}_j$ is a plain word, the symbol is concatenated with the decoded target phrase. If not, a pointer for the subphrase $\mathbf{s}'$ is reconstructed and the rank r' of the target phrase is determined. If the target subphrase $\mathbf{t}'$ has been decoded before, it is retrieved from the cache, else the encoded versions of $\mathbf{t}'$ and $\hat{A}'$ are fetched from the phrase table. $\mathbf{t}'$ and A' are obtained by recursively calling DecodeTargetPhrase. The decoded target subphrase is then concatenated with the current target phrase and subphrase alignment points are added to the output alignment, shifted accordingly. Results are cached before return.

4. Results

Coppa, the Corpus Of Parallel Patent Applications (Pouliquen and Mazenc, 2011) is used for phrase table generation. It comprises ca. 8.7 million parallel segments, 198.8 million English and 232.3 million French tokens. The generated phrase table consists of 247 million phrase pairs. All phrase table variants include alignment information, a requirement for several WIPO applications. Performance test were conducted on an Amazon EC2 server with 8 cores and 70 GB RAM. The first unique 1000 sentence pairs from the WIPO test set[1] are translated for performance tests using all

[1]`http://www.wipo.int/patentscope/translate/coppa/testset2011.tmx.tgz`

```
**Function** GetTargetPhraseCollection($\mathbf{s}$)
    $\mathbf{T} \leftarrow \langle\,\rangle$; $r \leftarrow 0$
    **while** $r <$ NumberOfTargetPhrases($\mathbf{s}$) **do**
        **if** InCache($\mathbf{s}, r$) **then**
            $\langle \mathbf{t}, A \rangle \leftarrow$ GetFromCache($\mathbf{s}, r$)
        **else**
            $\langle \hat{\mathbf{t}}, \hat{A} \rangle \leftarrow$ GetFromPhraseTable($\mathbf{s}, r$)
            $\langle \mathbf{t}, A \rangle \leftarrow$ DecodeTargetPhrase($\mathbf{s}, r, \hat{\mathbf{t}}, \hat{A}$)
        $\mathbf{T} \leftarrow \mathbf{T} \cdot \langle \langle \mathbf{t}, A \rangle \rangle$
        $r \leftarrow r + 1$
    **return** $\mathbf{T}$
**Function** DecodeTargetPhrase($\mathbf{s}, r, \hat{\mathbf{t}}, \hat{A}$)
    $\mathbf{t} \leftarrow \langle\,\rangle$; $A \leftarrow \hat{A}$
    $j \leftarrow 0$
    **while** $j < |\hat{\mathbf{t}}|$ **do**
        **if** Type($\hat{t}_j$) = Pointer **then**
            $\langle k, l, r' \rangle \leftarrow \hat{t}_j$
            $i \leftarrow k + |\mathbf{t}|$
            $m \leftarrow |\mathbf{s}| - l + 1$
            $\mathbf{s}' \leftarrow$ Substring($\mathbf{s}, i, m$)
            **if** InCache($\mathbf{s}', r'$) **then**
                $\langle \mathbf{t}', A' \rangle \leftarrow$ GetFromCache($\mathbf{s}', r'$)
            **else**
                $\langle \hat{\mathbf{t}}', \hat{A}' \rangle \leftarrow$ GetFromPhraseTable($\mathbf{s}', r'$)
                $\langle \mathbf{t}', A' \rangle \leftarrow$ DecodeTargetPhrase($\mathbf{s}', r', \hat{\mathbf{t}}', \hat{A}'$)
            $\mathbf{t} \leftarrow \mathbf{t} \cdot \mathbf{t}'$
            $A \leftarrow A \cup \{\langle i + i', j + j' \rangle : \langle i', j' \rangle \in A'\}$
        **else if** Type($\hat{t}_j$) = Word **then**
            $\mathbf{t} \leftarrow \mathbf{t} \cdot \langle \hat{t}_j \rangle$
        $j \leftarrow j + 1$
    AddToCache($\mathbf{s}, r, \langle \mathbf{t}, A \rangle$)
    **return** $\langle \mathbf{t}, A \rangle$
```

Figure 4. Retrieving a set of target phrases

cores. Results are reported in Table 1 for: translation model size (Files), peak resident memory usage (RSS), peak operation system cache usage (Cached), warm-up time (Load), translation time without warm-up (Trans.), and time until the first translation is produced after warm-up (First). Before za "1st run" the operation system I/O caches have been dropped. During the "2nd run" all I/O caches of the first run are available.

System name	Memory (GB)			1st run (s)			2nd run (s)		
	Files	RSS	Cached	Load	Trans.	First	Load	Trans.	First
Moses	22.01	2.54	3.48	14	251	9	0	151	9
Moses+LR	68.40	4.00	5.84	15	470	8	3	215	8
Compact	4.77	1.76	4.85	18	195	2	4	135	1
Compact+LR	6.19	2.02	5.84	17	330	4	4	192	1
PR-Enc	2.93	1.84	3.98	15	170	1	3	137	<1
PR-Enc+LR	4.36	2.11	4.69	15	256	2	3	199	<1
PR-Enc (mem)	2.93	4.49	4.16	51	137	<1	5	138	1
PR-Enc+LR (mem)	4.36	5.91	5.59	66	196	<1	8	198	<1

Table 1. Comparison of phrase table implementations

We compare the following configurations: the standard Moses binary phrase table ("Moses"), the compact phrase table from Junczys-Dowmunt (2012a) without any encoding methods ("Compact") and with PR-Enc ("PR-Enc"). Additionally, phrase tables are combined with corresponding implementations of lexical reordering models ("+LR"), i.e. the Moses phrase table is used with the Moses binary reordering table, the compact phrase table is combined with a reordering model based on the compact implementation. In-memory variants of the compact phrase and reordering tables are denoted by "(mem)". All systems use the same 3-gram KenLM (file size 1.1 GB), which is responsible for a part of the load time and memory usage.

To sum up the results in Table 1: during second runs, speed is nearly identical for both variants of the compact table — with or without PR-Enc — and always better than for the Moses binary tables. During first runs, however, we see how reduced file size and reduced disk access result in increased speed. The complex PR-Enc decompression procedure has only a minor influence on speed, visible during second runs. PR-Enc reduces the size of the compact phrase table by another 39 percent. Compared to the Moses phrase table, size reduction reaches an order of magnitude, for lexical reordering models even more (46.4 GB versus 1.4 GB). Memory requirements for on-disk access are also more modest. If the phrase table and reordering table are loaded into memory, RSS memory usage is higher than for the Moses tables, but read cache usage is comparable. Translation speed, however, is much better despite increased load time. For more than 1000 sentences, cache usage will keep increasing in case of the Moses binary tables, for the compact in-memory version, it remains constant.

5. Conclusions

We introduced Phrasal Rank Encoding, a new method for the compression of translation phrase tables. The size reduction and performance improvement compared to the Moses binary phrase table and a basic version of our phrase table are significant. Our implementation can be successfully used in place of the Moses binary phrase table. Experiments were performed for a medium sized phrase table and it is planned to repeat them with a phrase table that contains billions of phrase pairs. We suspect that increasing phrase table size might work to the advantage of our implementation, as should the usage of much weaker machines.

6. Usage

We include only basic instructions, see the Moses website for more information on compact phrase tables[2]nd lexical reordering tables[3]

Download and install the CMPH library[4], next recompile a current Moses version:

```
./bjam --with-cmph=/path/to/cmph
```

For PR-Enc, the phrase table should include alignment information. The following command creates a compact binary phrase table `phrase-table.minphr` from a standard text version with PR-Enc enabled by default:

```
mosesdecoder/bin/processPhraseTableMin -in phrase-table.gz \
-out phrase-table -use-alignment -threads 4
```

The compact phrase table variant without PR-Enc can be created by adding the option `-encoding None`. In the Moses config file, the filename stem `phrase-table` has to be specified and the type is to be set to 12, i.e.:

```
[ttable-file]
12 0 0 5 phrase-table
```

A compact lexical reordering model `reordering-table.minlexr` can be created with the following command:

```
mosesdecoder/bin/processLexicalTableMin -in reordering-table.gz \
-out reordering-table -threads 4
```

If only the file stem is given in the configuration file, the compact model is loaded instead of any other present lexical reordering model.

In-memory storage of the phrase table and the reordering model can be forced by running Moses with the options `-minphr-memory` and `-minlexr-memory` correspondingly. These can also be specified in the configuration file.

Acknowledgements

Part of this research was carried out at and funded by the World Intellectual Property Organization (WIPO) in Geneva.

[2]`http://www.statmt.org/moses/?n=Moses.AdvancedFeatures#ntoc5`

[3]`http://www.statmt.org/moses/?n=Moses.AdvancedFeatures#ntoc6`

[4]`http://sourceforge.net/projects/cmph/`

Bibliography

Anh, Vo Ngoc and Alistair Moffat. Index Compression using Fixed Binary Codewords. In *Proceedings of the 15th Australasian Database Conference*, volume 27, pages 61–67, 2004.

Callison-Burch, Chris, Colin Bannard, and Josh Schroeder. Scaling Phrase-Based Statistical Machine Translation to Larger Corpora and Longer Phrases. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 255–262, 2005.

Conley, Ehud S. and Shmuel T. Klein. Using Alignment for Multilingual Text Compression. *International Journal of Foundations of Computer Science*, 19(1):89–101, 2008.

Germann, Ulrich, Eric Joanis, and Samuel Larkin. Tightly Packed Tries: How to Fit Large Models into Memory, and Make them Load Fast, Too. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 31–39, 2009.

Huffman, David. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.

Johnson, J. Howard, Joel Martin, George Fost, and Roland Kuhn. Improving Translation Quality by Discarding Most of the Phrasetable. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 967–975, 2007.

Junczys-Dowmunt, Marcin. A Phrase Table without Phrases: Rank Encoding for Better Phrase Table Compression. In *Proceedings of the 16th Annual Conference of the European Association for Machine Translation*, pages 245–252, 2012a.

Junczys-Dowmunt, Marcin. A Space-Efficient Phrase Table Implementation Using Minimal Perfect Hash Functions. In *Proceedings of 15th International Conference on Text, Speech and Dialogue*, volume 7499 of *Lecture Notes in Computer Science*, pages 320–328. Springer Verlag, 2012b.

Knight, Kevin and Philipp Koehn. What's New in Statistical Machine Translation. Tutorial at the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2003.

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

Pouliquen, Bruno and Christophe Mazenc. COPPA, CLIR and TAPTA: Three Tools to Assist in Overcoming the Language Barrier at WIPO. In *Proceedings of Machine Translation Summit XIII*, 2011.

Sadat, Fatiha, Howard Johnson, Akakpo Agbago, George Foster, Joel Martin, and Aaron Tikuisis. Portage: A Phrase-based Machine Translation System. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pages 129–132, 2005.

Sanchez-Martinez, Felipe, Rafael C. Carrasco, Miguel A. Martinez-Prieto, and Joaquin Adiego. Generalized Biwords for Bitext Compression and Translation Spotting. *Journal of Artificial Intelligence Research*, pages 389–418, 2012.

Scholer, Falk, Hugh E. Williams, John Yiannis, and Justin Zobel. Compression of Inverted Indexes for Fast Query Evaluation. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229, 2002.

Zens, Richard and Hermann Ney. Efficient Phrase-table Representation for Machine Translation with Applications to Online MT and Speech Translation. In *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2007.

Zens, Richard, Franz Josef Och, and Hermann Ney. Phrase-Based Statistical Machine Translation. In *Proceedings of the 25th Annual German Conference on AI: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 18–32. Springer Verlag, 2002.

Ziv, Jacob and Abraham Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transaction on Information Theory*, 23(3):337–343, 1977.

Address for correspondence:
Marcin Junczys-Dowmunt
junczys@amu.edu.pl
Information Systems Laboratory,
Faculty of Mathematics and Computer Science,
Adam Mickiewicz University
ul. Umultowska 87
61-614 Poznań, Poland

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 75–85

TrTok: A Fast and Trainable Tokenizer for Natural Languages

Jiří Maršík[a], Ondřej Bojar[b]

[a] Shanghai Jiao Tong University
[b] Institute of Formal and Applied Linguistics, Charles University in Prague

Abstract

We present a universal data-driven tool for segmenting and tokenizing text. The presented tokenizer lets the user define where token and sentence boundaries should be considered. These instances are then judged by a classifier which is trained from provided tokenized data. The features passed to the classifier are also defined by the user making, e.g., the inclusion of abbreviation lists trivial. This level of customizability makes the tokenizer a versatile tool which we show is capable of sentence detection in English text as well as word segmentation in Chinese text. In the case of English sentence detection, the system outperforms previous methods. The software is available as an open-source project on GitHub[1].

1. Introduction

Researchers in statistical machine translation and other natural language processing fields make use of large corpora of text. However, not all of these corpora are immediately useful since not all of them are partitioned into words and sentences. This is in odds with the premise that words and sentences, not chunks of text, form the basic processing units of most NLP applications. This is where tokenization and segmentation have to step in.

Segmentation (a term we use for what is also referred to as sentence detection or sentence boundary disambiguation) has been tackled using a variety of techniques. The most common approaches include writing heuristics and constructing abbreviation lists (the Stanford Tokenizer, the RE system) or using machine learning algorithms to predict the role of a potential sentence terminator (Satz, MxTerminator,

[1]`https://github.com/jirkamarsik/trainable-tokenizer`

Corresponding author: bojar@ufal.mff.cuni.cz
Cite as: Jiří Maršík, Ondřej Bojar. TrTok: A Fast and Trainable Tokenizer for Natural Languages. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 75–85. doi: 10.2478/v10108-012-0010-0.

Apache OpenNLP). There have also recently been some very successful systems using unsupervised methods (Punkt).

Tokenization is a problem which stops being trivial when we start considering whitespace-free languages such as Chinese or Japanese. In these languages, tokenization (also referred to as word segmentation) receives a lot of attention (Emerson, 2005).

TrTok aims to be a practical tool for tokenizing and segmenting text written in any language. To achieve such a goal, TrTok relies on the user determining the specifics of training and tokenization, and providing the necessary training data.

Continuing the approach outlined by Klyueva and Bojar (2008), TrTok's novelty comes in the openness and formalization of the tokenization process and in its resulting general applicability. The process is divided into several discrete stages, most of which are heavily customizable. For example, the user is able to say where in the text TrTok should consider breaking up or joining tokens or sentences, how TrTok should represent the context of these decision points to the underlying classifier, how the classifier should be trained, how existing whitespace should be treated and more.

TrTok was also built to be a practical tool, which means it can transparently process text interspersed with XML tags and HTML entities and it was designed to run fast.

The major inconveniences of TrTok are that, 1) due to its customizability it needs to be properly set up and, 2) due to its reliance on machine learning methods, it requires manually tokenized training data.

2. Previous Work

Established methods of sentence boundary disambiguation can be organized into three distinct groups: rule-based systems, supervised learning systems and unsupervised learning systems.

The **RE** system (Silla and Kaestner, 2004) is an example of a rule-based system. The program scans a document, looking for full stops. When one is found, the word preceding it is compared to a list of regular expression exceptions (mostly abbreviations) and unless the word is found to match one of them, it is assumed to end the sentence. Besides this core logic, the system also implements a small heuristic which checks for numbers preceding the full stop and the word following it.

MxTerminator (Reynar and Ratnaparkhi, 1997) is a supervised sentence boundary disambiguator using maximum entropy models to predict whether a potential sentence terminator does indeed signal the end of a sentence. The prefix and suffix of the word containing the potential sentence terminator and the words preceding and following it are analyzed and their features are passed to the classifier. The features consist of the tokens' type, their capitalization and their membership status on a list of abbreviations which are either hand-prepared or induced from data.

The biggest difference between TrTok and MxTerminator is that TrTok does not assume any particular selection of features and thus offers space for richer models (e.g., by extending the width of the context or providing more complex features like part of speech tags).

An example of a system using more advanced features is the **Satz** system (Palmer and Hearst, 1997), which uses possible POS (part of speech) tags as features in the machine learning algorithm.

Unsupervised learning systems are the most distinct from TrTok amongst all the sentence boundary detection algorithms as they usually require no manual configuration nor any training data to function properly. A great example of an unsupervised sentence boundary disambiguator is the **Punkt** system (Kiss and Strunk, 2006).

Punkt relies mostly on collocation detection techniques but also makes use of an orthographic heuristic to analyze the test data in several passes and disambiguate abbreviations and sentence terminators. The system has shown remarkable performance without needing any manual tuning or training data.

3. System Description

TrTok is implemented by a parallel execution of several, configurable pipeline steps. This pipeline can be repurposed to train the embedded classifier using tokenized data, to tokenize new data using a trained classifier, and to evaluate the predictions of a trained classifier on manually tokenized data.

We will describe the important pipeline steps one by one, in the order in which they process data when tokenizing new text.

3.1. RoughTokenizer

The RoughTokenizer partitions the stream of input characters into small, discrete chunks of non-blank characters called *rough tokens*. The partitioning can be made more granular by user-defined rules which specify positions at which the desired tokenization might differ from the whitespace-induced one.

A location in the text may be marked as a `MAY_SPLIT` meaning that the characters in the text preceding and following it may be parts of different tokens even though they are not separated by whitespace (e.g. we might wish to put a `MAY_SPLIT` between *"was"* and *"n't"* in *"wasn't"*).

A location within a span of whitespace characters might be labeled as a `MAY_JOIN` signalling that the characters preceding and following the whitespace might be parts of the same token, as in the case of spaces entered in long numbers (e.g. *"12 345"*).

Finally, a location in the text may be marked as a `MAY_BREAK_SENTENCE` if the characters preceding and following it might belong to different sentences.

See Figure 1 for an example of how these potential tokenization operations can look like in a sentence. A rough token is defined as a maximal sequence of characters uninterrupted by whitespace nor by any symbol denoting a possible tokenization operation (the symbols underneath the sentence in Figure 1). For example, in the

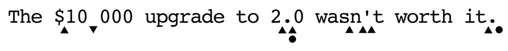


Figure 1. In this example, ▲ stands for MAY_SPLIT, ▼ for MAY_JOIN and ● for MAY_BREAK_SENTENCE. This is how the rough tokenization might turn out given some reasonable settings for tokenizing English.

sentence from Figure 1, "was", "n", the apostrophe and "t" are all individual rough tokens. Note that the presence of a `MAY_*` event only signifies the possibility of a tokenization operation (splitting or joining of tokens or sentences). Whether a token split, token join or sentence break will occur is up to the Classifier.

The locations of these possible tokenization operations are determined by user-defined rules, each of which consists of a pair of regular expressions. The respective tokenization operation is signalled if the characters leading up to and following a position match the regular expressions in one of these rules.

If we look back at Figure 1, we might imagine more robust settings also placing a `MAY_BREAK_SENTENCE` after the apostrophe/single quote, while others might be more daring and not place a `MAY_BREAK_SENTENCE` after the point in "2.0", because it is followed by a non-blank character.

TrTok collects these rules and generates a Quex program implementing a fast FSM (Quex[2] is a fast and Unicode-friendly variation on the classic tools lex and flex for C++).

3.2. FeatureExtractor

The stream of rough tokens interleaved with potential tokenization operations output by the RoughTokenizer is processed using the FeatureExtractor. The FeatureExtractor annotates each rough token with a bit vector signifying which of the user-defined feature predicates hold for the rough token in question.

The features can be defined in two ways: either using a regular expression or a list of rough tokens. For a feature defined using a regular expression, a rough token is said to have that feature if and only if the regular expression matches the entire rough token. In the case of a feature defined using a list of rough tokens, a rough token is said to have that feature if and only if it is in the list.

This way it is easy to specify features which try to analyze the shape of rough tokens using regular expressions or to simply give a list of all interesting tokens (e.g. words of a certain part of speech or exceptions such as abbreviations).

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

3.3. Classifier

The Classifier is the other important element in the pipeline (besides the RoughTokenizer). Its job is to disambiguate the potential tokenization operations identified by the RoughTokenizer, i.e. it decides whether a `MAY_SPLIT` splits a word into two tokens, whether a `MAY_JOIN` truly joins two words into one token and whether a `MAY_BREAK_SENTENCE` ends a sentence. It does so by consulting a maximum entropy classifier for every location containing these potential tokenization operations.

The features passed to the classifier consist of the features of rough tokens in the context surrounding the potential tokenization operation and the presence of whitespace and potential tokenization operations in the context area. The user is free to select the size of the context area and which features from which rough tokens in the context area should be passed to the classifier.

Features can also be clustered together into conjunct features which provide a value for every combination of the constituent features' values (this lets the trainer estimate a different parameter for different combinations of the features' values, which is useful to model the joint influence of some features).

The classifier then marks each location with a potential tokenization operation as either a sentence boundary, token boundary or no boundary (meaning the location is inside a token). Using this classification, any potential tokenization operations are finally disambiguated.

The model used in the Classifier unit is a maximum entropy model trained using the Maximum Entropy Modeling Toolkit for Python and C++[3]. Training is performed via either the L-BFGS or GIS algorithm, depending on the user's choice. Other parameters of the learner, such as the number of iterations to spend on training, are controlled by the user as well.

3.4. OutputFormatter

The OutputFormatter is the point at which the stream of rough tokens is turned back into a character stream. This means that all the rough tokens are concatenated and whitespace is inserted between them depending on whether there originally was any whitespace between them and on the tokenization operations which are to be carried out in the space between them. Individual tokens end up being separated by a single space character and sentences are separated by line breaks.

4. Usage

TrTok is used as a command line application.

```
Example:
   trtok train en/satz-like/brown -l data/brown/train.fl -r "|raw|txt|"
```

[3]`http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html`

Its first argument is the mode of operation, which can be one of `train`, `tokenize`, `evaluate` or `prepare`. The `train` mode uses manually annotated files to train a model for the Classifier and save it, while the `evaluate` mode uses them to compare the tokenizer's predictions to the manual tokenization and outputs the comparisons. The `tokenize` mode takes the input files and segments them using the trained model. The `prepare` mode does the same but with a dummy model which performs every possible sentence and token break.

The second argument to TrTok is the tokenization scheme folder. The tokenization scheme folder contains a set of optional files which influence the behavior of the tokenizer. Files with the `.rep` and `.listp` extensions define new features in terms of regular expressions or lists of types, respectively. Files with the `.split`, `.join` and `.break` extensions contain pairs of regular expressions which define possible token splits, token joins and sentence breaks, respectively. The `features` file defines which features are to be used from which rough tokens relative to the possible tokenization operation. The `maxent.params` file contains values for tuning the performance of the training algorithm. The scheme folder also allows a few other configuration files for convenience. An important thing to note is that the scheme folders can be nested and that the inner schemes inherit all the files of the outer scheme, unless they provide their own files of the same name. This is useful in cases where e.g. some features or training data are applicable to all texts of a language but refinements exist for various domains or tokenization conventions.

The rest of the parameters are input files and various options for adjusting the behavior of the tokenizer.

TrTok requires CMake and Quex at runtime, while several multi-platform libraries are also required at compile time. Further details on the installation and use of TrTok can be found in the bundled documentation.

5. Evaluation

We evaluated our implementation of TrTok compared to a wide range of prominent implementations and approaches to sentence detection. The results are given in Table 1.

5.1. Dataset

The experiments were conducted on the Brown corpus (Francis and Kucera, 1982) as supplied through NLTK (Bird et al., 2009). A representative (covering each category of text proportionately) 20% of the corpus was used as the testing data. This number was chosen so that the testing data would be sure to contain at least 1,000 instances of a non-sentence-terminating full stop; the resulting test set ended up containing 1,481 such full stops. The rest of the data was made available for training to the supervised learning methods.

	Acc. ↓	Err. Rate	Prec.	Recall	F_1	Time
TrTok::Groomed	**98.86%**	**1.14%**	**99.12%**	99.57%	**99.34%**	5.10s
Stanford CoreNLP	98.83%	1.17%	98.78%	99.89%	99.33%	5.02s
TrTok::MxTerm-like	98.76%	1.24%	98.70%	99.89%	99.29%	1.10s
TrTok::Easy	98.70%	1.30%	98.61%	99.91%	99.26%	1.08s
Punkt	98.65%	1.35%	98.82%	99.63%	99.22%	3.13s
MxTerminator	98.27%	1.73%	98.30%	99.74%	99.01%	1.37s
Apache OpenNLP	98.20%	1.80%	98.20%	99.77%	98.97%	1.13s
Apache OpenNLP (ready)	97.71%	2.29%	98.62%	98.75%	98.68%	1.17s
RE	97.26%	2.74%	98.52%	98.32%	98.42%	16.93s
TrTok::Satz-like	96.50%	3.50%	97.91%	98.08%	97.99%	1.59s
TrTok::Baseline	91.84%	8.16%	91.67%	99.66%	95.50%	0.85s
Absolute Baseline	86.89%	13.11%	86.89%	**100.00%**	92.99%	**0.02s**

Table 1. The performance of the various sentence detectors on full stops from the Brown corpus testing data. The 1.15 MB of testing data consisted of 11,376 sentences and 232,893 tokens.

5.2. Performance Measurement

The performance of the evaluated systems was measured by their success (accuracy) in classifying instances of the full stop character. The text contains other sentence terminators such as the question mark and the exclamation mark, but they almost never serve as anything else but sentence terminators in the text. Other occasional sentence delimiters such as dashes, semicolons, colons and line breaks were ignored as well, since the other systems usually do not have a solution for them. This way, the comparison is fair. Furthermore, the full stop is the most common and ambiguous of the sentence delimiters, so it makes sense to focus on it.

Besides the systems' accuracy, we also measure the time spent for processing the whole testing data and we present the median of 11 runs to bring the implementation speed of the systems into consideration as well.

5.3. Sentence Detection Methods

Absolute Baseline simply marks every full stop as a sentence terminator.

Trtok::Baseline is the simplest tokenizer which can be written in TrTok. However, even the simplest TrTok configuration always uses the whitespace following the possible tokenization operation as a feature and thus it is able to perform better than the Absolute Baseline.

TrTok::Satz-like is a straightforward attempt at reconstructing the Satz system in TrTok. The POS-tagged training data was used to construct lexicons for each different part of speech tag (NLTK's method of simplifying tags was used to reduce the number of different tags to overcome data sparsity). The POS tags for three tokens on either side of the full stop were used as the features.

TrTok::Satz-like's system of tags is not as refined as the original and it does not use its fallback regular expression heuristics and hence it does not perform as well as the original Satz system did (Palmer and Hearst, 1997).

The **RE** system, **MxTerminator** and **Punkt** were described in Section 2. For training, Punkt received the entire Brown corpus (training data and testing data) without any annotations while MxTerminator was trained using the training data.

Punkt achieves remarkable performance and stands as the strongest competitor to TrTok in the field of multilingual sentence detection. They are both accurate language-independent tools but TrTok's big shortcoming is its need for a corpus of manually tokenized data.

Apache OpenNLP contains a sentence detector based around a maximum entropy classifier. The implementation is nearly identical to the specification of MxTerminator with only minor deviations (such as signalling surrounding whitespace as features).

We performed experiments both with the ready-made model for English distributed via OpenNLP's website and with a model which was trained on our training data.

The **Stanford CoreNLP** sentence splitter works by applying its tokenizer to the input text which makes the distinction between a full stop as part of an abbreviation or an ordinal number as opposed to a full stop as a sentence terminator. Thus the task of sentence splitting is trivial after the tokenization has been performed. The tokenizer is a rule-based program implemented using a lexical analyzer generator, JFlex (similar to how TrTok uses Quex to implement the RoughTokenizer).

The Stanford Tokenizer's performance is excellent, especially considering it has not had the chance to train itself on the target corpus. However, the Stanford Tokenizer is written explicitly for English and it is likely that its performance would not carry over to other languages without significant work.

TrTok::MxTerm-like is a reconstruction of MxTerminator in TrTok. It is a nice demonstration of the ease with which new tokenization setups can be defined in TrTok. The entire setup consisted of creating a directory, collecting the abbreviations in a single file and writing five lines of configuration, two or three of which could be easily obsoleted by adopting saner defaults in TrTok and one of which is purely for convenience.

The reason why MxTerminator does not achieve the same performance could be that the maximum entropy trainer used in MxTerminator limits itself to 100 iterations of Generalized Iterative Scaling, which converges very slowly compared to L-BFGS (Malouf, 2002). Another reason might be the fact that both MxTerminator and OpenNLP cut off infrequent features.

The high accuracy of TrTok::MxTerm-like led us to try and see what happens when we simplify the tokenization setup even further, which led to **TrTok::Easy** which works the same way as TrTok::MxTerm-like, but which does not use any abbreviation lists, merely the token types surrounding the full stop. Therefore, TrTok::Easy

	True Words Recall	Test Words Precision	F-measure
Academia Sinica	0.933	0.919	0.926
City University	0.934	0.934	0.934
Peking University	0.923	0.933	0.928
Microsoft Research	0.951	0.952	0.951

Table 2. The scores assigned to our tokenizer by the official scoring script of the Second International Chinese Word Segmentation Bakeoff, sorted by dataset.

does not rely on any external linguistic knowledge and is fairly language universal, given that we have enough training data. The performance of TrTok::Easy also points out that the difference in performance between TrTok and MxTerminator/OpenNLP cannot be explained by the different abbreviation lists.

Finally, **TrTok::Groomed** is a large, hand-made tokenization setup ported from a previous version of the tokenizer. It considers 24 different potential sentence terminators, it includes seven distinct lists of abbreviations totalling 303 types (prefix and suffix titles, abbreviated names of months, etc.) and it implements features for detecting the case of tokens, for noticing numbers which happen to be in the range of years, or the days of the month, etc… These features are extracted from rough tokens within eight tokens distance from the full stop. The two closest tokens on either side of the full stop also contribute their token type as a feature.

Due to the large number of potential tokenization operations and user-defined features, TrTok::Groomed's speed lags significantly behind the other TrTok setups.

Interestingly, there is not much difference in the performance of TrTok::Groomed, TrTok::MxTerm-like and TrTok::Easy. This tells us that besides the token types in the close vicinity of the full stop, other features are not that important. This highlights another use for TrTok as a tool for the fast analysis of the importance of different contextual features for performing the task of sentence detection.

5.4. Chinese Word Segmentation

Since TrTok is a general program for splitting text into sequences (sentences) which are in turn composed of other sequences (tokens) based on user-defined features, TrTok can be used for more than just sentence detection. One other segmentation task we had hoped might be solvable using TrTok is Chinese word segmentation.

We ported the key features of one of the top contestants (which also happens to employ a maximum entropy classifier) (Low et al., 2005) in the 2005 Second International Chinese Word Segmentation Bakeoff into TrTok and evaluated its performance using the official evaluation scripts. The results achieved (see Table 2) are approximately a median of the scores reported for submissions to the Bakeoff.

6. Conclusion

We have presented and described a universal tool for segmenting and tokenizing textual data. We have applied the tool to detecting sentences in English text and identifying words in Chinese text. We have shown that in both cases, TrTok can offer performance which is competitive with previous approaches, more so in the case of English sentence detection. In our experiments, different setups of TrTok outperformed existing systems in either speed or accuracy, while some setups of TrTok outperformed nearly all competitors in both criteria at the same time.

Since TrTok lets us define a lot of its behavior using declarative rules and feature descriptions, it might be interesting to harness this ability to find out the effect of various contextual cues on the performance of a sentence detector.

On the software side of things, TrTok would also benefit from getting more user-friendly, which would entail providing a walkthrough of the setup process, distributing further example setups and trained models and offering an all-dependencies-included compiled package for easier deployment.

Bibliography

Bird, S., E. Klein, and E. Loper. *Natural language processing with Python*. O'Reilly Media, 2009.

Emerson, T. The second international chinese word segmentation bakeoff. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, volume 133. Jeju Island, Korea, 2005.

Francis, W. and H. Kucera. Frequency analysis of english usage. 1982.

Kiss, T. and J. Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

Klyueva, N. and O. Bojar. Umc 0.1: Czech-russian-english multilingual corpus. In *Proc. of International Conference Corpus Linguistics*, pages 188–195, 2008.

Low, J.K., H.T. Ng, and W. Guo. A maximum entropy approach to chinese word segmentation. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, volume 1612164. Jeju Island, Korea, 2005.

Malouf, R. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

Palmer, D.D. and M.A. Hearst. Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics*, 23(2):241–267, 1997.

Reynar, J.C. and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19. Association for Computational Linguistics, 1997.

Silla, C.N. and C.A.A. Kaestner. An analysis of sentence boundary detection systems for english and portuguese documents. *Lecture notes in computer science*, pages 135–141, 2004.

Address for correspondence:
Ondřej Bojar
`bojar@ufal.mff.cuni.cz`
Institute of Formal and Applied Linguistics
Charles University in Prague
Malostranské náměstí 25
118 00 Praha 1, Czech Republic

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 87–98

Parallel Phrase Scoring for Extra-large Corpora

Mohammed Mediani, Jan Niehues, Alex Waibel

Karlsruhe Institute für Technology

Abstract

This paper presents a C++ implementation of the phrase scoring step in phrase-based systems that helps to exploit the available computing resources more efficiently and trains very large systems in reasonable time without sacrificing the system's performance in terms of Bleu score.

Three parallelizing tools are made freely available. The first exploits shared memory parallelism and multiple disks for parallel IOs while the two others run in a distributed environment.

We demonstrate the efficiency and consistency of our tools, in the framework of the Fr-En systems we developed for the WMT and IWSLT evaluation campaigns, in which we were able to generate the phrase table in one third up to one seventh of the time taken by *Moses* in the same tasks.

1. Introduction

Phrase scoring is one of the most important and yet very expensive steps in phrase-based translation system training. Typically, it consists of estimating the corresponding scores for each unique phrase pair extracted from an aligned parallel corpus. Usually, the scores are estimated based on two directions (from source to target and vice versa). Therefore, the process is accomplished in two runs. In the first run, counts are collected and then the scores are estimated based on the source phrases while in the second run a similar task is performed based on the target phrases.

This process is memory greedy. However, for non large corpora it could be performed efficiently in the physical memory by some implementations. For instance, *memscore* (Hardmeier, 2010) uses a lookup hash table based on STL[1] maps to index

[1]C++ Standard Template Library `http://www.sgi.com/tech/stl/`

Corresponding author: mohammed.mediani@kit.edu

the phrases. Then the hash identifiers are used to directly access the corresponding phrases in order to update the marginal and joint counts. Unfortunately, this does not scale very well for corpora of large sizes. As a matter of fact, a memory requirement of more than 60GiB was reported for a corpus of 4.7M sentence pairs (Hardmeier, 2010).

On the other hand, most systems such as the widely used phrase-based system *Moses* (Koehn et al., 2007), handle the memory limitation by streaming the large data sets, keeping only a limited amount of data into memory, and saving temporary results into disk. In fact, all the pairs which correspond to a given phrase should be kept into memory while gathering the marginal and joint counts for this phrase. Consequently, the streamed data must be sorted depending on whether the computation is being held based on source phrases or target phrases . In *Moses*, this is achieved by performing two sorting operations using the standard Unix *sort* command.[2] Even though, being a good external memory sorting tool, the Unix *sort* command is not optimal when the corpus is very large. For instance, the runs are formed and sorted serially, it lacks support for multiple disks, and the IO could not be overlapped with the computations.

Gao and Vogel (2010) developed a platform for distributed training of phrase-based systems starting from word alignment until phrase scoring. Even though excellent speed gains were reported, this system runs on top of the Hadoop framework, and therefore needs the cluster to fit this special infrastructure.

Unlike applications which operate exclusively on data stored in main memory, applications which involve external memories such as hard disks face an additional challenge with the high data transfer latency between the external and main memory. For this purpose, data structures and algorithms have been developed in order to minimize the IO overhead and to exploit the available resources such as parallel disks and multiple processors more efficiently (Vitter, 2008). Luckily, different external memory APIs have been created in order to make the underlying disk access and low level operations transparent to programmers. Such platforms include, but are not limited to, LEDA-SM (Crauser and Mehlhorn, 1999), TPIE (Arge et al., 2002), Berkeley DB (Olson et al., 1999), and STXXL (Dementiev and Kettner, 2005).

The main goals of our tools for phrase scoring are to exploit CPU and disk parallelism in an external memory environment, so that the phrase sorting and score computation are performed more efficiently. The CPU parallelism is ensured by the OpenMP library (Chapman et al., 2007) (eventually coupled with an MPI implementation (Pacheco, 1996)), while the disk parallelism and other external memory functionalties are ensured by the STXXL library. STXXL is preferred over other environments due to its superior performance, ease of use (STL-compatible interface), and explicit support for parallel disks (Dementiev et al., 2008).

Most of our tools are written in C++. The underlying CPU parallelism comes in three flavours: multithreaded, hybrid, and distributed. The multithreaded version

[2]`http://unixhelp.ed.ac.uk/CGI/man-cgi?sort`

uses shared memory parallelism and therefore runs on a single node. In the hybrid setting, multiple nodes can be used. On each of these nodes the shared memory parallelism is exploited. The distributed tool proceeds in a MapReduce strategy (Dean and Ghemawat, 2008): Starting from Giza alignments, the large corpus is split into partitions and training is performed independently on the partitions. For each part, standard *Moses* tools are used for alignment combination, lexical scoring, and phrase extraction. For phrase scoring, we use a slightly modified version of the multithreaded tool. It allows all the partial counts to be saved as well. The partial phrase tables are then merged and the probabilities are reestimated using the new updated counts.

In the next section, the external memory sorting is briefly presented in the framework of the STXXL implementation. Afterwards, the architecture and underlying algorithms of our different software flavours are dissected and its usage is explained. Then some experimental results are presented and discussed. Finally, a conclusion about the main findings and eventual extensions ends the paper.

2. External memory sorting in STXXL

Due to its extreme importance, the external memory sorting has received continuous improvements over the years. The different techniques can be categorized in two classes: distribution sorts and merging sorts. A detailed survey of both approaches can be found in Vitter (2008).

Details about STXXL sort implementation are given in Sanders and Dementiev (2003). In the following, we briefly review its important aspects.

STXXL implements a multiway-merge sort. It assumes that the data records are of fixed size. The processing then could be held on fixed size data blocks. The STXXL library forms the backbone of many sorting benchmark[3] winners in the past years (Andreas et al., 2011; Rahn et al., 2009; Beckmann et al., 2012). The two key steps of STXXL sorting are as follows:

Run formation In a double buffering strategy, two threads cooperate to read/sort the different runs. The first thread sorts the run which occupies half of the sorting memory, while the second thread is either reading the next run or writing the sorted run. The sorter thread creates lighter data structure consisting of only the keys and pointers to the actual elements. After that, it sorts the keys in the new data structure where the sorting method depends on their number (straight line code if it doesn't exceed 4, insertion sort if it is between 5 and 16, otherwise it uses quicksort).

Multiway merging In order to define the order in which blocks will be streamed into the merger, the smallest elements in each block are recorded in a sorted list during run

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

formation. The position of an element in this list defines when its containing block will enter the *merging buffers*. The merger keeps a number of blocks equal to the number of the sorted runs in merging buffers. In order to minimize the time of selecting the current smallest element, the keys of the smallest elements of all blocks in merging buffers are kept in a tree sctructure.

STXXL uses an *overlap buffer* for reading and a *write buffer* for writing in order to overlap IOs and merging. The size of the overlap buffer depends both on the number of runs and the number of parallel disks while the size of the write buffer depends on the number of disks only. If the write buffer has a number of blocks which exceeds the number of disks, a parallel output is submitted. Similarly, if the overlap buffer has a number of free blocks which exceeds the number of disks, a parallel read is performed.

Distributed External Memory sorting (DEMSort) is an extension of the STXXL sorting so that it fits the distributed case where the sorting is rather performed on multiple machines (Rahn et al., 2010). The key difference here is the introduction of an additional intermediate phase between run formation and multiway merging: the so-called *Multiway selection*.

Like the distribution sorts, the multiway selection tends to find global splitting points over all the sorted runs. By the end of this operation, each node knows its exclusive range of data. Afterwards, the data are redistributed globally over the nodes using an all-to-all operation to satisfy the range constraints. In this case, the MPI interface is used for the inter-node communication. Finally, the merging is done locally as explained before.

3. Software architecture and algorithms

Like *Moses* scoring tool, our phrase scoring tools take three files as input and produce a phrase table as output. The first input file contains the extracted phrases (called 'extract.0-0.gz' in Moses convention) and the other files are two bilingual dictionaries which model $\Pr(s \mid t)$ and $\Pr(t \mid s)$ for every source and target words s and t if they are aligned at least once ('lex.0-0.f2e' and 'lex.0-0.e2f' in Moses convention).

Typically, the phrase table records 4 scores for every extracted phrase pair. Relative frequency and lexical score for each direction (source to target and vice versa). Our lexical score is identical to the one produced by Moses Scoring tool, whereas our relative frequency is smoothed using modified Kneser-Ney smoothing as described in Foster et al. (2006).

The development of our tools led to three different levels of parallelism: multithreaded, hybrid, and distributed. The multithreaded version forms the core of the other two versions. The multithreaded and hybrid versions parallelize only the phrase scoring whereas the distributed version parallelizes the former steps too. In the following, we explain each of these versions.

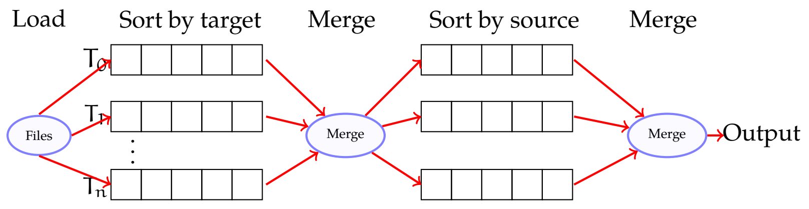


Figure 1. Multithreaded phrase scoring anatomy

3.1. Multithreaded phrase scoring

The basic data structure used in this software is STXXL vector whose interface is similar to STL vector but it rather stores data which does not all reside in memory. STXXL vector elements are stored in the form of key, value. The keys of this vector are the phrase pairs (source and target phrases concatenated) and the values are the different counts. In order to satisfy the fixed size record of STXXL vectors, the keys are represented by a fixed-length string.

As depicted in Figure 1, the process consists of several threads, each of which takes care of one large STXXL vector of data. The phrase table is the result of five consecutive steps. Details about each of these steps are presented in what follows.

Loading the data First of all, the lexical dictionaries are loaded into two STL maps (one for each direction). Afterwards, each thread reads one phrase pair at a time, computes its lexical score, and then loads it into its corresponding STXXL vector. This multithreaded way allows for computations and IOs to be overlapped.

There are two ways to read pairs from the file into memory. The fast way: where all the threads read the same file concurrently one line at a time. In this case, the input file should not be zipped. The alternative way allows to read directly from the zipped file, the master reads from the file and pushes the lines into a FIFO queue. The other threads pop lines from the queue and process them.

As soon as the loading is complete, the lexical maps are disposed since they will not be needed anymore.

Sorting by target phrases Every thread sorts its vector by simply calling the STXXL sort function which performs a multiway merging sort on the corresponding vector.

Merging and computing the target-based scores The merging follows the same approach as the multiway merging. The first elements from all vectors are organized in a tree structure. Whenever an element is taken out, it is replaced with the next element from the same vector.

Parallel threads acquire a lock on the tree and get all the pairs with the same target phrase in a local vector, then release the lock for the next thread. After collecting the pairs, every thread updates the corresponding count fields and writes the updated records to a new STXXL vector. Since the identical pairs have to be uniquified in this step, our implementation allows chosing one lexical score and one alignment based on maximal lexical score or the most occurring one.

Sorting by source phrases Again, this is done in parallel by the STXXL sort.

Merging and computing the source-based scores This operation is identical to the merge based on target phrases.

Writing out the phrase table Like the loading phase, two writing ways are possible. The way which supports writing zipped phrase table is performed by a single thread while the multithreaded way writes only unzipped files.

Optionally, all the counts can be recorded for further use (as in the distributed version). It is as well possible to write out an optional abridged phrase table containing only phrases which match a list of given n-grams.

3.2. Hybrid parallel phrase scoring

The extension DEMSort allows us to effieciently sort an STXXL vector spread over multiple interconnected machines. There are only few changes in the architecture compared to the previous version. We suppose that the nodes dispose of a shared disk space. First of all, all the nodes build the lexical maps in the same way. Afterwards, every node reads a quota of the input file of phrase pairs into an STXXL vector. Running the DEMSort could raise the following issue: the phrase pairs which correspond to a given phrase could be spread between two adjacent nodes due to the redistribution as explained in Section 2. To fix this, every node sends all the phrase pairs corresponding to the first phrase to its immediate predecessor. As a consequence of this sorting approach, no further data exchange between the nodes is needed.

Every node performs the local merging and scoring strictly identical to the multithreaded version. In our development process, this resulted in an unbalanced load between the nodes. Consequently, we extended the merging with a dynamic load balancing strategy. The final merging procedure executed on every node looks as follows:

1. Execute a multithreaded merging and listen to signals from other nodes
2. If request for sharing is received from another node, then send half of the remaining pairs to that node
3. When finished, signal all other nodes
4. If all nodes have no remaining pairs, then exit

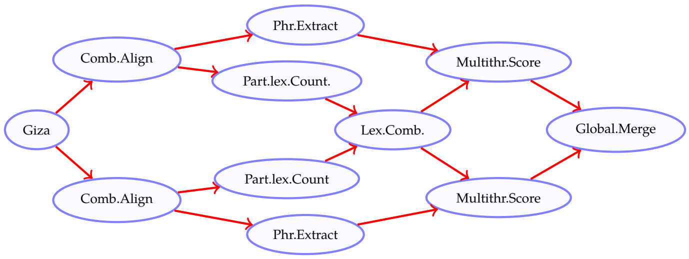


Figure 2. Distributed phrase table construction process (2 partitions)

5. Receive half of the remaining pairs from the node with the largest remaining number of pairs
6. Go to 1

The output is done in a similar manner to the previous system where all the nodes write to the same file concurrently. The position from which a node starts writing in the common output file is estimated based on the number of enteries in this node's vector.

3.3. Distributed translation model construction

This version is based on two complementary pieces: the aforementioned multithreaded scorer and a multithreaded merger. The objective of the latter is to merge streamed partial phrase tables produced by the scorer.

In fact, starting from partitioned Giza alignement files all the subsequent steps are run independently (typically on a cluster of machines). However, a slight modification is introduced in this pipeline in order to produce correct lexical dictionaries.The counts for aligned words are collected from each part independently and then globally combined in an additional step from all the collected counts. The sequencing and dependence between the different steps of this version is shown in Figure 2 (for a number of partitions equals to 2).

The global merger is very similar in design to the multithreaded scorer. The only difference is that the counts are not initialized to 0, but rather based on the saved counts in the partial phrase tables. Afterwards, it proceeds in the same steps as the multithreaded scorer.

3.4. Usage

All our tools show different options by specifying `-h` or `--help` flag.

Multithreaded phrase scorer The number of threads can be specified by setting the `OMP_NUM_THREADS` environment variable. If this variable is not set, it will be set to the number of physical cores available on the machine. The most important options and flags for this software are (all options can be printed using the `-h` flag):

`-e, -l, -L` are used to provide the extract, lexical dictionary source to target, lexical dictionary target to source files respectively.

`-b` with this option, the sorting internal memory per thread can be specified in Mega bytes.

`-w` is used to specify the different disks (paths) which will serve as parallel disks for STXXL sort. It is a comma-separated list.

Hybrid phrase scorer The binary in this case is called `pscore`. It accepts the same arguments as the previous one. Though, it needs to be started with `mpirun`.

Running the distributed version on a cluster The script which automates the partitioning the Giza alignment files and ensures the correct dependency between the jobs (as shown in Figure 2) is written in Python and uses specific commands for the *Slurm*[4] queue manager. We believe however that it can be easily adapted to other schedulers.

4. Experimental results

In this section, we show some performance comparisons between the different versions of our scoring tools. We compare them as well to Moses. The hardware environment where these experiments took place is a cluster consisting of 8 core machines with 32GiB of memory and 16 core machines with 64GiB memory.

All the machines have access to a RAID NFS shared space and dispose of a local disk of 1.7TiB. In all experiments the parallel scorers use two disks for the STXXL vectors (the local disk and NFS). The first set of experiments (in WMT2011) was held on the 8 core machines, while the others were held on the 16 core machines.

Experiments in the WMT2011 In this set of experiments, the Multithreaded version was run on a 16-core machine, whereas the hybrid was run on four different machines (using 4 cores out of 8 on each one). Table 1 compares the speed of different tools used in this experiment, whereas Table 2 shows the Bleu scores resulting from a system based on Moses phrase table and the hybrid balanced system (we kept only one phrase table since all the tables produced by our tools are identical). These phrase tables are trained based on three parallel corpora (merged into a single large corpus): EPPS, NC, and UN. The total number of parallel sentences is 13.8 millions. Clearly, the best choice here is the hybrid balanced version. It is 7.5 times faster than Moses

[4] `https://computing.llnl.gov/linux/slurm/`

scorer. However, explicitly handling the communication (for both versions) and the load balancing (for the balanced version) from whithin the scoring routines degrades readability, and thus maintaining this code became expensive. This was the reason why we next created the fully distributed version and we didn't report further tests with the hybrid version.

System	Time span
Moses	53h 34m
Multithreaded	28h 49m
Hyb. unbalanced	8h 45m
Hyb. balanced	7h 08m

Table 1. Phrase scoring time span in WMT2011

System	en-fr	fr-en
Moses	23.16	24.16
Parallel scoring	23.24	24.21

Table 2. Bleu scores in WMT2011

System	EPPS+NC	+UN
Moses	11h 23m/27.21	49h 34m/29.13
Multithr.	9h 34m/27.5	27h 44m/29.02

Table 3. Phrase scoring in IWSLT2011

System	Time	Bleu
Moses	92h 46m	29.77
Distributed	49h 20m	30.00

Table 4. Phrase scoring in WMT2012

Experiments in the IWSLT2011 Experiments in this context are shown for Moses vs. the multithreaded version for the same corpora as the previous. For every corpus and system, Table 3 gives the corresponding time span and Bleu score. As in the previous experiment, the amount of speed up becomes more and more appearent as the corpus size augments, while the translation model's performance in terms of Bleu scores is almost invariant. However, the slight difference (Table 3, column +UN compared to Table 1) is mainly due to a different set of disks.

Experiments in the WMT2012 This set of experiments is held between Moses and the distributed version. In addition to the EPPS, NC, and UN corpora, the training data here include the Giga corpus as well (resulting in 29.4 millions parallel sentences). The number of partitions here was 12 and the jobs were submitted independently to the cluster (some of them end up on the same node, which is not optimal). Table 4 records the time required for phrase scoring. It is shown here that the distributed version is almost 2 times faster than Moses.

It is noteworthy that relative frequency in Moses version here was also modified as in Foster et al. (2006). These experiments show that not only our tools are faster than Moses, but they also produce in most cases slightly better results. We think the reason for that is due to the lexical score selection explained in Section 3.1, unlike Moses where the first occurring score is selected.

Surprisingly, the distributed version was not as fast as the hybrid version. This could be justified by the race condition which occurs during concurrent access to the the NFS space by so many processes.

5. Conclusion

In this paper, we presented three versions of a tool which makes the phrase scoring manageable for extra large corpora. This was achieved by exploiting multiple processing units and parallel disk IOs using the STXXL library for external memories. The first implementation can be run on a single machine. Whereas the other two can be executed in a multinode environment (typically on a cluster of nodes). The three implementations are freely available under the LGPL license and can be downloaded from `http://isl-wiki.ira.uka.de/~mmediani/fscorer`. All these tools depend on the STXXL and OpenMP libraries. In addition to that, the hybrid version assumes the existence of an MPI implementation and the DEMSort extension for the STXXL library.

Given that the bottleneck in this process is the slow disk speeds compared to internal memory, the amount of improvement strongly depends on the number of parallel disks. This could be shown by the experiment in Section 4, where the hybrid version performed better than other versions since it uses multiple nodes each of which uses its local disk as well as the NFS space. The distributed version is still being tested and optimized, therefore the speedup it brings is still low compared to the hybrid version even though they are somehow similar in spirit.

Since the objective of the experiments shown in this paper was to participate in the MT evaluation campaign, they were run on relatively powerful hardware. However, these tools would also work for less powerful architectures, since the memory consumption is bounded by design.

The main limitation of our tools is the disk space consumption. This is essentially due to the fact that our basic data structure uses a fixed size character string for the keys of our STXXL vectors. As a result, some very long pairs cannot be taken into account and shorter ones have to be filled with blank characters. This implies that a considerable amount of the space allocated for keys is not useful. A possible solution to this would be to use suffix arrays to index the phrases and use only the ID's in the STXXL vector keys.

Acknowledgements

This work was realized as part of the Quaero Programme, funded by OSEO, French State agency for innovation. The authors are also glad to Yuqi Zhang for her comments and suggestions on this paper.

Bibliography

Andreas, Beckmann, Meyer Ulrich, Sanders Peter, and Singler Johannes. Energy-efficient sorting using solid state disks. *Sustainable Computing: Informatics and Systems*, 1(2):151–163, 2011.

Arge, Lars, Octavian Procopiuc, and Jeffrey Scott Vitter. Implementing I/O-efficient data structures using TPIE. In *In Proc. European Symposium on Algorithms*, pages 88–100. Springer, 2002.

Beckmann, Andreas, Ulrich Meyer, Peter Sanders Johannes Singler, and Peter Sanders Johannes Singler. Energy-efficient fast sorting 2011, 2012. URL `http://sortbenchmark.org/demsort_2011.pdf`.

Chapman, Barbara, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

Crauser, Andreas and Kurt Mehlhorn. LEDA-SM extending LEDA to secondary memory. In *Proceedings of the 3rd International Workshop on Algorithm Engineering*, WAE '99, pages 228–242, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66427-0.

Dean, Jeffrey and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008. ISSN 0001-0782.

Dementiev, R. and L. Kettner. STXXL: Standard template library for XXL data sets. In *In: Proc. of ESA 2005. Volume 3669 of LNCS*, pages 640–651. Springer, 2005.

Dementiev, R., L. Kettner, and P. Sanders. STXXL: standard template library for XXL data sets. *Softw. Pract. Exper.*, 38(6):589–637, May 2008. ISSN 0038-0644.

Foster, George F., Roland Kuhn, and Howard Johnson. Phrasetable smoothing for statistical machine translation. In *EMNLP*, pages 53–61, 2006.

Gao, Qin and Stephan Vogel. Training phrase-based machine translation models on the cloudopen source machine translation toolkit chaski. *Prague Bull. Math. Linguistics*, 93: 37–46, 2010.

Hardmeier, Christian. Fast and extensible phrase scoring for statistical machine translation. *Prague Bull. Math. Linguistics*, 93:87–96, 2010.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

Olson, Michael A., Keith Bostic, and Margo Seltzer. Berkeley DB. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '99, pages 43–43, Berkeley, CA, USA, 1999. USENIX Association.

Pacheco, Peter S. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. ISBN 1-55860-339-5.

Rahn, Mirko, Peter S, Johannes Singler, and Tim Kieritz. DEMSort-distributed external memory sort, 2009. URL `http://sortbenchmark.org/demsort.pdf`.

Rahn, Mirko, Peter Sanders, and Johannes Singler. Scalable distributed-memory external sorting. In on Data Engineering (ICDE), International Conference, editor, *26th IEEE International Conference on Data Engineering, March 1-6, 2010, Long Beach, California, USA*, pages 685–688. IEEE Computer Society, März 2010.

Sanders, Peter and Roman Dementiev. Asynchronous parallel disk sorting. Research Report MPI-I-2003-1-001, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, February 2003.

Vitter, Jeffrey Scott. *Algorithms and Data Structures for External Memory*. Now Publishers Inc., Hanover, MA, USA, 2008. ISBN 1601981066, 9781601981066.

Address for correspondence:
Mohammed Mediani
`mohammed.mediani@kit.edu`
Karlsruhe Institute für Technology
Adenauerring 2,
Karlsruhe 76131, Germany

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 99–108

rgbF: An Open Source Tool for n-gram Based Automatic Evaluation of Machine Translation Output

Maja Popović

DFKI, Language Technology Group

Abstract

We describe RGBF, a tool for automatic evaluation of machine translation output based on n-gram precision and recall. The tool calculates the F-score averaged on all n-grams of an arbitrary set of distinct units such as words, morphemes, POS tags, etc. The arithmetic mean is used for n-gram averaging. As input, the tool requires reference translation(s) and hypothesis, both containing the same combination of units. The default output is the document level 4-gram F-score of the desired unit combination. The scores at the sentence level can be obtained on demand, as well as precision and/or recall scores, separate unit scores and separate n-gram scores. In addition, weights can be introduced both for n-grams and for units, as well as the desired n-gram order n.

1. Motivation

Evaluation of machine translation output is an important but difficult task. A number of automatic evaluation measures have been studied over the years, some of them have become widely used by machine translation researchers, such as the BLEU metric (Papineni et al., 2002) and the Translation Edit Distance TER (Snover et al., 2006). Precision and recall are used for machine translation evaluation in Melamed et al. (2003) and it is shown that they correlate well with human judgments, even better than the BLEU score. Recent investigations have shown that the n-gram based evaluation metrics BLEU and F-score calculated on Part-of-Speech (POS) sequences correlate very well with human judgments (Popović and Ney, 2009) clearly outperforming the widely used metrics BLEU and TER. However, using only POS tags for evaluation has

Corresponding author: maja.popovic@dfki.de
Cite as: Maja Popović. rgbF: An Open Source Tool for n-gram Based Automatic Evaluation of Machine Translation Output. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 99–108.
doi: 10.2478/v10108-012-0012-y.

certain disadvantages, for example the translation hypotheses "The flowers are beautiful" and "The results are good" would have the same score. Therefore combining lexical and non-lexical "units", e.g. words and POS tags seemed to be a promising direction for further investigation.

The RGBF tool presented in this work enables calculation of such combined scores, i.e. F-score of an arbitrary combination of distinct units (words, POS tags, morphemes, etc). The tool has been successfully used in the sixth WMT evaluation shared task (Popović, 2011; Callison-Burch et al., 2011), and it is confirmed that introducing the morphological and syntactic properties of involved languages thus abstracting away from word surface particularities (such as vocabulary and domain) improves the correlation with human judgments, especially for the translation from English.

The name RGBF refers to the RGB color model used in computer graphics: in this model, primary colors red, green, and blue are added together in various ways thus producing a broad array of different colors. Our evaluation tool adds together individual scores for different basic units and n-gram orders in various ways thus producing a broad array of evaluation scores. The final letter F stands for the F-score which is the default output.

The tool is written in Python, and it is available under an open-source licence. We hope that the release of the toolkit will facilitate the automatic evaluation for the researchers, and also stimulate further development of the proposed method.

2. RGBF tool

2.1. Algorithm

RGBF implements the precision, recall and F-score of all n-grams up to order n of all desired units. The arithmetic averaging of n-grams is performed – previous experiments on the syntax-oriented n-gram metrics (Popović and Ney, 2009) showed that there is no significant difference between arithmetic and geometric mean in the terms of correlation coefficients. In addition, it is also argued that the geometric mean used for the BLEU score is not optimal because the score becomes equal to zero even if only one of the n-gram counts is equal to zero, which is especially problematic for the sentence level evaluation.

The recall is defined as percentage of words in the reference which also appear in the hypothesis, and analogously, the precision is the percentage of words in the hypothesis which also appear in the reference. Multiple counting is not allowed. For example, for the hypothesis "this is a hypothesis and this is a hypothesis" and the reference "this is a reference and this is a hypothesis" the unigram precision will be 8/9=88.9% and not 9/9=100%. In the case of multiple references, the highest precision and the highest recall score is chosen for each sentence (the optimal reference for the precision and the optimal reference for the recall are not necessarily the same). Once the recall and precision are obtained, the F-score is calculated as their harmonic mean.

Although the method is generally language-independent, availability of some kind of analyser for the particular target language might be required depending on which units are desired.

2.2. Usage

RGBF supports the option `-h/--help` which outputs a description of the available command line options.

The input options are:

`-R, --ref`	translation reference
`-H, --hyp`	translation hypothesis
`-n, --ngram`	n-gram order (default: $n = 4$)
`-uw, --uweight`	unit weights (default: $1/U$)
`-nw, --nweight`	n-gram weights (default: $1/n$)

Inputs `-R` and `-H` are required, containing an arbitrary number of different types of units. The combination of units must be the same and in the same order both in the reference and in the hypothesis, and the units must be separated by "++". This symbol is of course not needed if the input files contain only one unit. The required format for all input files is a raw tokenized text containing one sentence per line. In the case of multiple references, all available reference sentences must be separated by the symbol #.

The output options are:

- standard output – the default output of the tool is the overall (document level) 4-gram F-score.

In addition to the standard output, the following optional outputs are available:

`-p, --prec`	precision
`-r, --rec`	recall
`-u, --unit`	separate unit scores
`-g, --gram`	separate n-gram scores
`-s, --sent`	sentence level scores

An example of input and output files and different program calls is shown in the next section.

2.3. Example

Table 1 presents an example of translation hypothesis consisting of two sentences and its corresponding reference translation in the RGBF format. Both hypothesis and reference contain four types of units, i.e. full words, base forms, morphemes and POS tags, separated by ”++”.

example.hyp.wbmp (word+base+morph+pos)
This time , the reason for the collapse on Wall Street . ++ This time , the reason for the collapse on Wall Street . ++ Th is time , the reason for the collapse on Wall Street . ++ DT NN , DT NN IN DT NN IN NP NP SENT
The proper functioning of the market and a price . ++ The proper functioning of the market and a price . ++ The proper function ing of the market and a price . ++ DT JJ NN IN DT NN CC DT NN SENT

example.ref.wbmp (word+base+morph+pos)
This time the fall in stocks on Wall Street is responsible for the drop . ++ This time the fall in stock on Wall Street be responsible for the drop . ++ Th is time the fall in stock s on Wall Street is responsible for the drop . ++ DT NN DT NN IN NNS IN NP NP VBZ JJ IN DT NN SENT
The proper functioning of the market environment and the decrease in prices . ++ The proper functioning of the market environment and the decrease in price . ++ The proper function ing of the market environment and the decrease in price s . ++ DT JJ NN IN DT NN NN CC DT NN IN NNS SENT

Table 1. Example of a hypothesis and a corresponding reference containing four units: full words, base forms, morphemes and POS *tags merged in the* RGBF *format.*

1) *Simple program call* without optional parameters:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp
```

will calculate the document level F-score with the default n-gram order $n = 4$ and the uniform distribution of weights, i.e. all the n-gram weights are $1/n = 1/4 = 0.25$ and all the unit weights are $1/U$ where U is the number of different units ($U = 4$ for the input files presented in Table 1). The obtained output will be:

```
rgbF    42.2512
```

2) A *desired unit and/or n-gram weight distribution* can be demanded with a call:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -uw 2-3-4-6 -nw 2-2-5-5
```

where uw represents the proportion of unit weights and nw the proportion of n-gram weights. The weights are normalized automatically, so that the given numbers do not have to sum to 1, only to represent the desired proportion. The output of this call will be:

```
rgbF    36.5530
```

3) The weights also enable *the choice of units and/or n-grams*. For example, the call:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -uw 2-0-0-3
```

will produce the word+POS F-score averaged on unigrams, bigrams, trigrams and fourgrams in proportion 2 words : 3 POS, and the call:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -nw 1-0-0-1
```

will average over all units but only over unigrams and fourgrams.

4) A *desired maximum n-gram order* can also be demanded, for example 6-gram:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -n 6
```

5) *Precision and/or recall scores* can be requested:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -p -r
```

These scores will be then showed in addition to the default F-score:

```
rgbF      42.2512
rgbPrec   48.9473
rgbRec    37.1839
```

6) If *the sentence scores* are desired:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -s
```

the F-score of each sentence together with the sentence number will be showed in addition to the default document level F-score:

```
1::rgbF   31.0037
2::rgbF   55.8205
rgbF      42.2512
```

7) If *the unit scores* are demanded:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -u
```

the F-score of each unit will be showed in addition to the default overall F-score:

```
u1-F   36.6824
u2-F   38.7693
u3-F   40.2712
u4-F   53.2818
rgbF   42.2512
```

where the unit number is its position in the reference and hypothesis file. For our example, u1 stands for the full words, u2 for base forms, u3 are morphemes and u4 are POS tags.

8) Separate n-*gram scores* can also be demanded:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -g
```

so that the F-score of each n-gram of each unit will be showed in addition to the default overall F-score:

```
u1-1gram-F   68.0000
u1-2gram-F   39.1304
u1-3gram-F   23.8095
u1-4gram-F   15.7895
u2-1gram-F   72.0000
u2-2gram-F   43.4783
...          ...
u4-3gram-F   42.8571
u4-4gram-F   21.0526
rgbF         42.2512
```

9) The *most "complicated" program call* involving *all optional output parameters*:

```
rgbF.py -R example.ref.wbmp -H example.hyp.wbmp -p -r -u -g -s
```

will produce all the F-scores, precisions and recalls for each unit n-gram and each unit, on the sentence level and on the document level.

3. Correlations with human ranking

As mentioned in Section 1, the tool has been tested on all WMT data from year 2008 to year 2011. In addition, it has also been tested on the data developed in the framework of the TARAXÜ project[1]. Spearman's rank correlation coefficients ρ are calculated for the document (system) level correlation, whereas Kendall's τ coefficients are calculated for the sentence level correlation.

3.1. WMT data

The following 4-gram RGBF scores have been investigated on the WMT data: WORDF, MORPHF, POSF, WPF, WMF, MPF, as well as WMPF without and with given weights (WMPF'). Spearman's rank correlation coefficients on the document (system) level between all the metrics and the human ranking are computed on the English, French, Spanish, German and Czech texts generated by various translation systems in the framework of the third, fourth and fifth shared translation tasks (Callison-Burch et al., 2008, 2009, 2010), and the results are shown in Table 2.

[1]http://taraxu.dfki.de/

metric	overall	x→en	en→x
BLEU	0.566	0.587	0.544
WORDF	0.550	0.592	0.504
MORPHF	0.608	0.671	0.541
POSF	**0.673**	**0.726**	**0.617**
WPF	0.627	0.698	0.553
WMF	0.587	0.655	0.514
MPF	**0.669**	**0.744**	**0.590**
WMPF	0.645	0.721	0.565
WMPF′	**0.668**	**0.744**	**0.587**

Table 2. Average document level correlations on the WMT *2008–2010 data for the* BLEU *score and the investigated* RGB *metrics. Bold represents the best value in the particular metric group (single unit, two-unit and three-unit). The most promising metrics are those containing* POS *and morpheme information, namely* WMPF′ *(*WMPF *with non-uniform weights),* MPF *and* POSF.

The most promising metrics, i.e. MPF and WMPF′ are submitted to the sixth shared evaluation task (Callison-Burch et al., 2011) and the correlations on the document and on the sentence level are presented in Table 3, together with the widely used BLEU and TER metrics and the best ranked metrics MTERATERPLUS, TINE-SRL-MATCH, TESLA-F, TESLA-B, METEOR-adq, METEOR-rank and AMBER.

On the document level, the RGBF scores are better than BLEU and TER and comparable with the best ranked metrics for translation from English, however worse than the best ranked metrics for translation into English. On the sentence level, the RGBF scores are comparable with the best ranked metrics for translation into English, and one of the best for translation from English.

3.2. TARAXÜ data

The TARAXÜ corpora consist of two domains: News taken from the WMT 2010 News test set and technical documentation extracted from the freely available OpenOffice project (Tiedemann, 2009). The translation outputs are produced by four different German-to-English, English-to-German and Spanish-to-German machine translation systems: Google, Moses (statistical systems), Lucy (a rule-based system) and Trados (not really a system but a translation memory). The obtained outputs are then given to the professional human annotators to assign 1–4 ranks, but without ties. More details can be found in (Avramidis et al., 2012).

The following 4-gram RGB scores have been explored on this data: WORDF, BASEF, MORPHF, POSF, WPF, BPF, MPF, WMPF, MBPF and WMBPF, all with the default uniform weights.

metric	document level x→en	document level en→x	sentence level x→en	sentence level en→x
MPF	0.77	0.78	0.28	0.26
WMPF	0.76	0.77	0.27	0.25
BLEU	0.69	0.70	/	/
TER	0.67	0.57	/	/
MTeRaterPlus	0.90	/	0.37	/
TINE-SRL-MATCH	0.87	/	0.23	/
TESLA-F	0.86	0.94*	0.31	0.26*
TESLA-B	0.84	0.87*	0.30	0.25*
METEOR-adq	0.83	/	0.28	/
METEOR-rank	0.82	0.63	0.29	0.23
AMBER	0.80	0.70	0.27	0.26

*Table 3. Average document level and sentence level correlations on WMT 2011 shared evaluation task for two submitted RGB metrics, widely used BLEU and TER scores, and best ranked novel evaluation metrics. The results marked with * are averaged without the Czech translation outputs.*

Document level Spearman's coefficients and sentence level Kendall's coefficients are calculated for the BLEU score and for all investigated RGBF scores on all data, as well as separately for each language pair and for each domain.

On the document level no significant differences are observed – all the correlation coefficients are very high, between 0.8 and 1. Sentence level correlations are shown in Table 4. The results are similar to those on WMT data, i.e. most promising metric is the MPF score, followed by the WMPF and MBPF scores. Combining full forms and base forms of the words (WMBPF) does not yield any improvements.

4. Conclusions

We presented RGBF, a toolkit for automatic evaluation of translation output which we believe will be of value to the machine translation community. It can be downloaded from `http://www.dfki.de/~mapo02/rgbF/`.

So far, the most promising RGBF scores are those using morphemes and POS tags as units. Different unit and n-gram weights should be investigated in future work, as well as the use of other types of units.

	overall	de-en	en-de	es-de	news	openoffice
BLEU	-0.198	0.024	-0.250	-0.296	-0.181	-0.328
WORDF	0.557	0.592	0.544	0.544	0.549	0.619
BASEF	0.561	0.589	0.554	0.548	0.553	0.618
MORPHF	0.587	0.616	0.570	0.583	0.581	0.639
POSF	0.534	0.569	0.511	0.529	0.528	0.582
WPF	0.577	0.610	0.564	0.565	0.571	0.624
BPF	0.577	0.611	0.563	0.566	0.571	0.622
MPF	**0.597**	**0.623**	0.587	**0.589**	**0.591**	0.644
WMPF	0.595	0.622	0.582	0.587	0.588	0.645
MBPF	0.596	0.620	**0.589**	0.588	0.589	**0.654**
WMBPF	0.593	0.618	0.583	0.586	0.586	0.650

Table 4. Sentence level correlations on TARAXÜ data for the BLEU score and the investigated RGB metrics. Bold represents the best values. The most promising metrics are MPF, WMPF and MBPF.

Acknowledgments

This work has partly been developed within the TARAXÜ project financed by TSB Technologiestiftung Berlin – Zukunftsfonds Berlin, co-financed by the European Union – European fund for regional development.

Bibliography

Avramidis, Eleftherios, Aljoscha Burchardt, Christian Federmann, Maja Popović, Cindy Tscherwinka, and David Vilar. Involving language professionals in the evaluation of machine translation. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC12)*, Istanbul, Turkey, May 2012.

Callison-Burch, Chris, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. Further meta-evaluation of machine translation. In *Proceedings of the Third Workshop on Statistical Machine Translation (WMT 2008)*, pages 70–106, Columbus, Ohio, June 2008.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, and Josh Schroeder. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation (WMT 2009)*, pages 1–28, Athens, Greece, March 2009.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, Kay Peterson, Mark Przybocki, and Omar Zaidan. Findings of the 2010 Joint Workshop on Statistical Machine Translation and Metrics for Machine Translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR (WMT 2010)*, pages 17–53, Uppsala, Sweden, July 2010.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, and Omar Zaidan. Findings of the 2011 Workshop on Statistical Machine Translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation (WMT 2011)*, pages 22–64, Edinburgh, Scotland, July 2011.

Melamed, I. Dan, Ryan Green, and Joseph P. Turian. Precision and Recall of Machine Translation. In *Proceedings of the Human Language Technology Conference (HLT-NAACL 03)*, pages 61–63, Edmonton, Canada, May/June 2003.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wie-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 02)*, pages 311–318, Philadelphia, PA, July 2002.

Popović, Maja. Morphemes and POS tags for n-gram based evaluation metrics. In *Proceedings of the Sixth Workshop on Statistical Machine Translation (WMT 2011)*, pages 104–107, Edinburgh, Scotland, July 2011.

Popović, Maja and Hermann Ney. Syntax-oriented evaluation measures for machine translation output. In *Proceedings of the 4th EACL 09 Workshop on Statistical Machine Translation (WMT 2009)*, pages 29–32, Athens, Greece, March 2009.

Snover, Matthew, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A Study of Translation Error Rate with Targeted Human Annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 06)*, pages 223–231, Boston, MA, August 2006.

Tiedemann, Jorg. News from OPUS – A Collection of Multilingual Parallel Corpora with Tools and Interfaces. In *Recent Advances in Natural Language Processing*, volume V, pages 237–248. John Benjamins Amsterdam, Borovets, Bulgaria, 2009.

Address for correspondence:
Maja Popović
`maja.popovic@dfki.de`
German Research Center for Artificial Intelligence (DFKI)
Language Technology Group (LT)
Alt-Moabit 91c
10559 Berlin, Germany

PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 98 OCTOBER 2012 109–119

Better Splitting Algorithms for Parallel Corpus Processing

Lane Schwartz

Air Force Research Laboratory
Human Effectiveness Directorate
Wright-Patterson AFB, OH USA

Abstract

Each iteration of minimum error rate training involves re-translating a development set. Distributing this work across computational nodes can speed up translation time, but in practice some parts may take much longer to complete than others, leading to computational slack time. To address this problem, we develop three novel algorithms for distributing translation tasks in a parallel computing environment, drawing on research in parallel machine scheduling. We present results showing a substantial speedup in overall decoding time.

1. Introduction

The task of translation involves translating a source language document $\mathbf{f}$ into target language $\boldsymbol{e}$. Most popular statistical translation techniques select the best translation $\hat{e}$ for source sentence f according to a linear combination of models $\boldsymbol{\phi}$ using a set of model weights $\boldsymbol{\lambda}$ (Och and Ney, 2002).

$$\hat{e} = \arg\max_{e} \sum_{i} \lambda_i \phi_i(e, f) \tag{1}$$

Values for $\boldsymbol{\lambda}$ are obtained by optimizing an objective function such as BLEU (Papineni et al., 2001) against a development set, most commonly using minimum error rate training (MERT) (Och, 2003). Each iteration of MERT requires this development set to be re-translated using a new set of $\boldsymbol{\lambda}$ weights. MERT is one of the slowest components in a typical machine translation training pipeline, and translating the development set is nearly always the slowest step in MERT. We now examine techniques

Corresponding author: Lane.Schwartz@wpafb.af.mil
Cite as: Lane Schwartz. Better Splitting Algorithms for Parallel Corpus Processing. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 109–119. doi: 10.2478/v10108-012-0013-x.

for speeding up the translation process by splitting a source document into parts and distributing the translation of those parts across parallel computational nodes.

Ideally, all parts should take the same amount of time to translate. While naive splitting techniques reduce the time required for each translation iteration by splitting the work between n computational nodes, in practice some parts may take much longer to complete than others. This can lead to significant computational slack time. To address this problem, we develop three novel algorithms for splitting translation tasks in a parallel computing environment, drawing on research in parallel machine scheduling.

2. Related Work

Research into parallel machine scheduling problems constitutes a wide and well-studied field, ranging through various disciplines of engineering, manufacturing, and management in addition to computer science and applied mathematics (Cheng and Sin, 1999), spanning a wide range of scheduling techniques (Panwalkar and Iskander, 1977).

We now briefly examine the existing research most relevant to our task. Hu (1961) and Graham (1966; 1969) develop various list scheduling algorithms. This family of algorithms prioritizes jobs into a queue, then assigns jobs to machines in queue order. This approach attempts to evenly balance the load on each execution host (De and Morton, 1980; Cheng and Sin, 1999). Both Algorithm 1 below and the techniques we develop in Section 3 fall into this family of algorithms.

Algorithm 1 Split input text into n parts such that each part contains the same number of lines. In cases where the total number of lines is not evenly divisible by n, the last part will contain fewer lines than each of the other parts.

```
function NAIVE-SPLIT(n,input)
    $s \leftarrow$ input.length
    $\ell \leftarrow \lceil s/n \rceil$
    for $p \leftarrow 0 \ldots (n-1)$ do
        $i \leftarrow \ell \times p$
        for $j \leftarrow i \ldots \min(i+\ell-1, s-1)$ do
            output[p].append(input[j])
        end for
    end for
    return output
end function
```

While the models in Equation 1 could, in theory, condition on previously translated sentences, in practice virtually no widely used models do so. It is therefore very

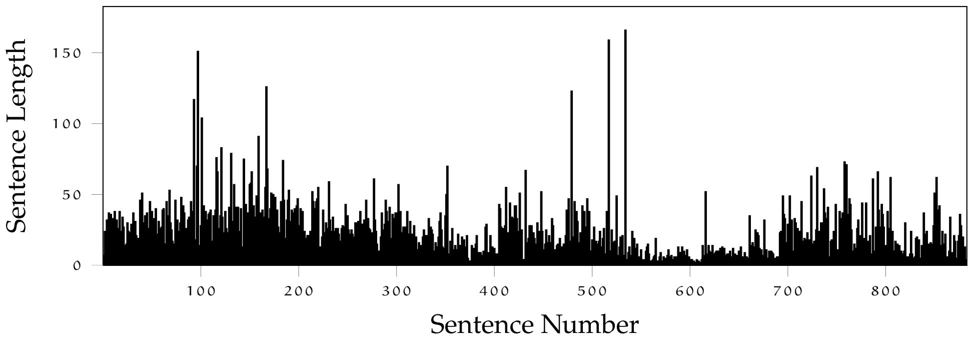


Figure 1: Sentence length (in words) of sentences from the NIST OpenMT 2008 Urdu-English task.

straightforward to split the data into n parts, and translate each part independently on n computational nodes.

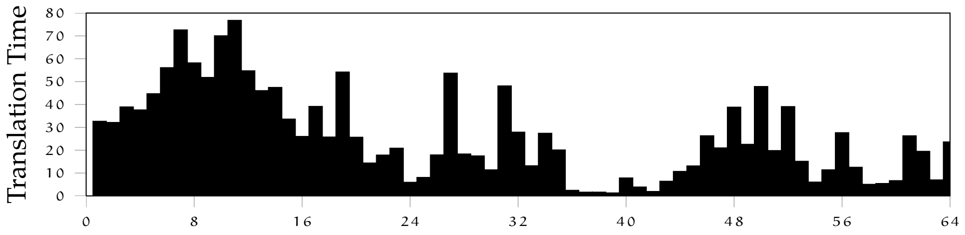
3. Better Splitting Algorithms

We begin by examining the development set for the NIST OpenMT 2008 Urdu-English task. When tuning the model weights λ, this development set will be translated numerous times. We observe in Figure 1 that the number of words in each sentence varies widely and unevenly throughout the corpus.

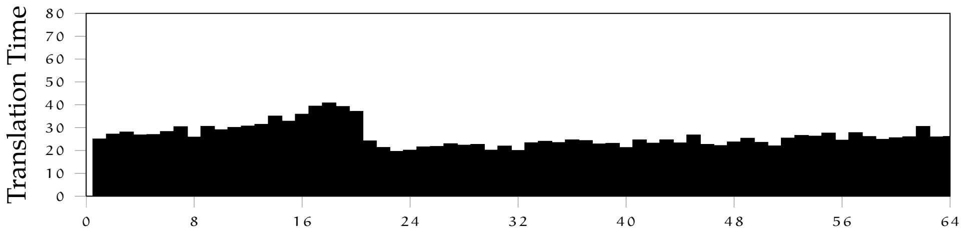
Using Algorithm 1, scripts included with Moses (Koehn et al., 2007) split the corpus into n parts of of ℓ or fewer lines each; ℓ is the smallest integer greater than or equal to s/n, where s is the total number of input sentences. The first ℓ lines comprise the first part, the next ℓ lines comprise the second part, and so on. Thus, each part contains exactly ℓ lines, with the possible exception of the last part, which contains fewer than ℓ lines when s is not evenly divisible by n.

Figure 2a shows the amount of time taken to translate each part of the development set, split using Algorithm 1 into 64 parts, using Moses configured with a 5-gram language model. We observe that the shape of Figure 2a generally matches that of Figure 1. There is substantial variance in translation time between the parts in Figure 2a, with some parts taking nearly 80 seconds and many others finishing in well under 10 seconds. We observe that this disparity is largely due to an imbalance of short versus long sentences between the parts. Because short sentences take less time to translate than long sentences, parts assigned mostly short sentences finish much faster than parts that are assigned many longer sentences.

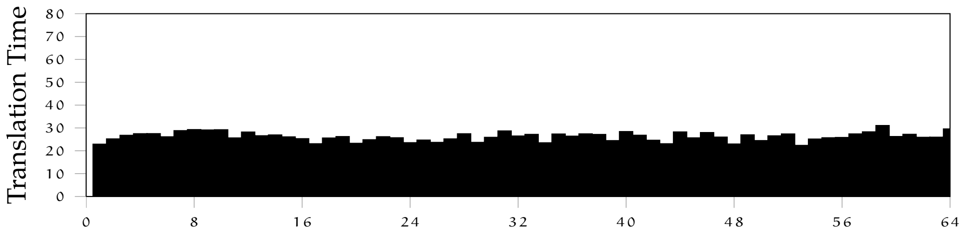
To remedy this imbalance, we propose Algorithm 2. Prior to splitting the data into parts, Algorithm 2 begins by examining the number of words in each sentence.



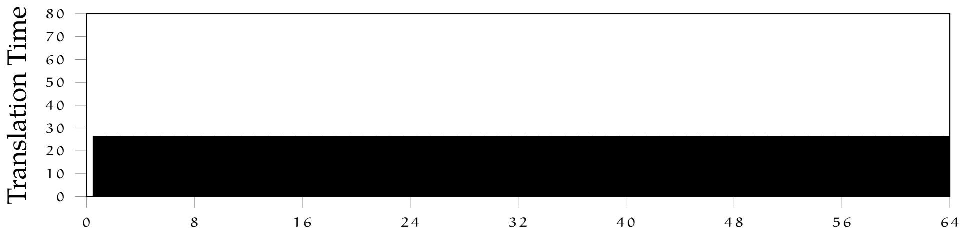
(a) Algorithm 1 — Naive-Split



(b) Algorithm 2 — Histogram-Split



(c) Algorithm 3 — Words-Split



(d) Algorithm 4 — Times-Split

Figure 2: Total translation time (in **seconds**) for each part when data from the NIST OpenMT 2008 Urdu-English task is split into 64 parts using each of four algorithms.

Algorithm 2 Split input text into n parts to balance the histograms of line lengths for all parts.

```
function HISTOGRAM-SPLIT($n$,input)
    for $i \leftarrow 0 \ldots$ ( input.length $-1$ ) do
        sentence[$i$].length $\leftarrow$ input[$i$].length
        sentence[$i$].index $\leftarrow i$
    end for
    SORT(sentence) {$|x, y|$ $x$.length $\Leftrightarrow$ $y$.length}
                                                    ▷ Sort sentences by length
    $p \leftarrow n$
    for $i \leftarrow 0 \ldots$ (input.length $-1$) do
        if $p < n - 1$ then
            $p \leftarrow p + 1$
        else
            $p \leftarrow 0$
        end if
        output[$p$].append(input[sentence[$i$].index])
    end for
    return output
end function
```

Sentences are sorted according to length, then assigned in turns to parts. This round-robin distribution of sentences into parts results in the sentence length histograms for each part being approximately equal. While Algorithm 2 attempts to balance short and long sentences across parts, we nevertheless observe non-trivial imbalance in translation times across parts in Figure 2b.

To improve this remaining imbalance, we propose Algorithm 3. In Algorithm 3, sentences are sorted by length into a queue, with longest sentences at the head of the queue. Initially, no sentences have been assigned to any part. The longest sentence, at the head of the queue, is assigned first to a part. As each sentence is assigned to a part, the total number of words assigned to that part is recorded. Each subsequent sentence is removed from the queue and assigned to the part with the least work assigned to it, as measured by number of words. In Figure 2c, we observe that most of the imbalance in translation times across parts has been resolved.

When assigning sentences to jobs, we would ideally like to know how long each sentence will take to process. Algorithms 2 and 3 use the number of words in each sentence as a proxy for processing time. During MERT, the same set of development sentences are translated multiple times. Since each decoding process differs only by the λ weights used, it is reasonable to expect little variation in decoding runtime for any given sentence across all MERT runs. With this in mind, we record the time required to translate each sentence during the first iteration of MERT. In subsequent

Algorithm 3 Split input text into n parts to balance the number of words for all parts.

```
function Words-Split($n$,input)
    for $i \leftarrow 0 \ldots$ ( input.length $-1$ ) do
        sentence[$i$].length $\leftarrow$ input[$i$].length
        sentence[$i$].index $\leftarrow i$
    end for
    Sort(sentence) $\{|x, y|\ y$.length $\Leftrightarrow x$.length$\}$
                                        ▷ Sort sentences by length, in reverse order
    for $i \leftarrow 0 \ldots$ ( input.length $-1$ ) do
        $p \leftarrow$ Least(words)
                                        ▷ Find partition with fewest words
        output[$p$].append(input[sentence[$i$].index])
        words[$p$] $\leftarrow$words[$p$] + sentence[$i$].length
    end for
    return output
end function
```

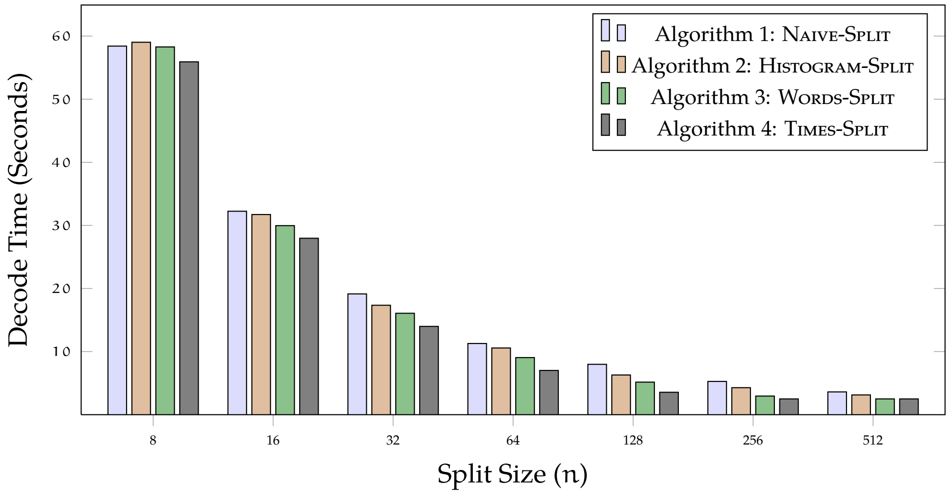
iterations, Algorithm 4 uses the time recorded to translate a sentence as an estimate of the time it will take to translate that sentence again. Algorithm 4 differs from Algorithm 3 by sorting using these times instead of sentence length. We see in Figure 2d that the time imbalance between parts is virtually non-existent, with all times now within 0.01 seconds of each other.

4. Experimental Configuration

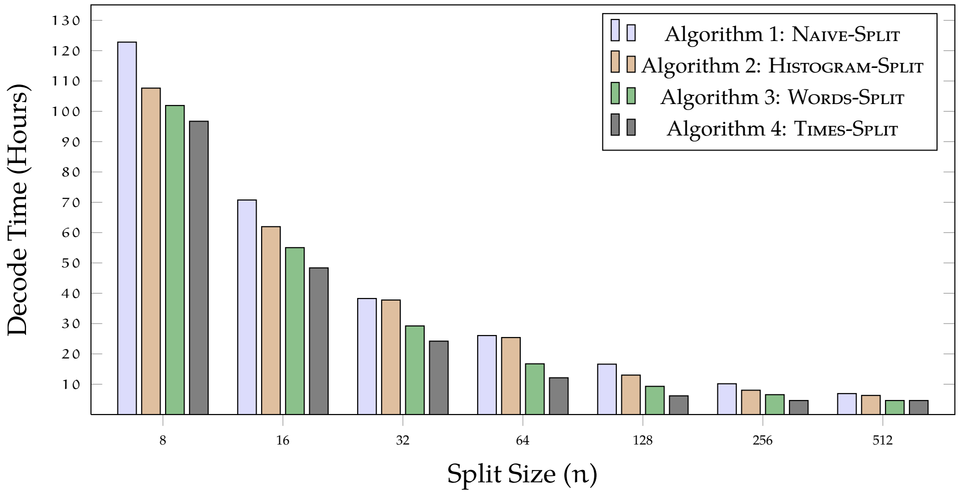
To observe the effects of splitting algorithms on decoding speed, we translated the NIST OpenMT 2008 development set of Urdu-English data using Moses in a parallel computing cluster, distributing work using the Sun Grid Engine. We ran two decoding setups: a standard configuration using a 5-gram language model, and a much slower syntactic LM configuration following Schwartz et al. (2011).

Figure 2 shows the per-part translation times for all parts of the development set when $n = 64$. In Figure 3, we examine the per-part translation times for only the slowest of n translation jobs in each configuration for various values of n, ranging from 2–512. In all configurations, we see that Algorithm 4 provides the fastest performance.

Another metric to use in examining our algorithms is total computational slack time. During MERT, computational slack time arises when some parts of the development set finish translating faster than others. Figure 4 lists the decoding times of the fastest and slowest translation jobs for parts split using each of the four algorithms for various values of n, ranging from 2–512. We see the total cumulative slack time for each of these conditions in Figure 5.



(a) Decoding times in **seconds** for decoder configured using a 5-gram language model.



(b) Decoding times in **hours** for decoder configured using a syntactic language model (Schwartz et al., 2011) in addition to a 5-gram language model.

Figure 3: Decoding times for the slowest translation job in a translation task split into n decoding jobs using various splitting algorithms (Naive-Split, Histogram-Split, Words-Split, and Times-Split).

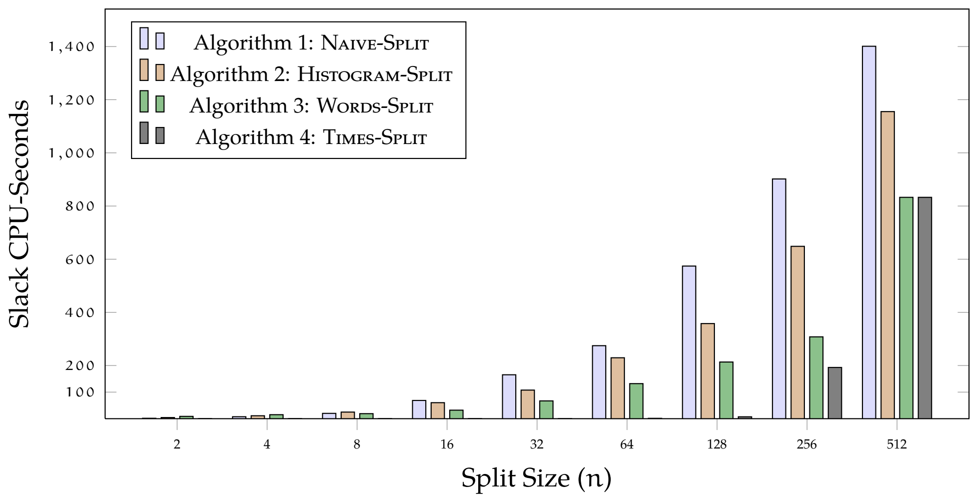
Split Size	Algorithm 1 Naive-Split		Algorithm 2 Histogram-Split		Algorithm 3 Words-Split		Algorithm 4 Times-Split	
(n)	Min	Max	Min	Max	Min	Max	Min	Max
2	222.9	224.4	221.5	225.8	219.3	228.0	*223.7*	***223.7***
4	109.2	113.7	110.0	114.6	108.7	115.6	*111.8*	***111.8***
8	51.4	58.4	52.2	59.0	53.2	58.3	*55.9*	***55.9***
16	24.9	32.2	25.0	31.7	25.3	30.0	27.9	**28.0**
32	11.3	19.1	11.9	17.3	11.7	16.1	*14.0*	***14.0***
64	5.4	11.3	5.4	10.6	5.7	9.1	*7.0*	***7.0***
128	1.3	8.0	2.2	6.3	2.3	5.2	*3.5*	***3.5***
256	0.3	5.3	0.7	4.3	0.8	3.0	1.7	**2.5**
512	0.0	3.6	0.2	3.1	0.3	**2.5**	0.6	**2.5**

(a) Decoding times in **seconds** for decoder configured using a 5-gram language model.

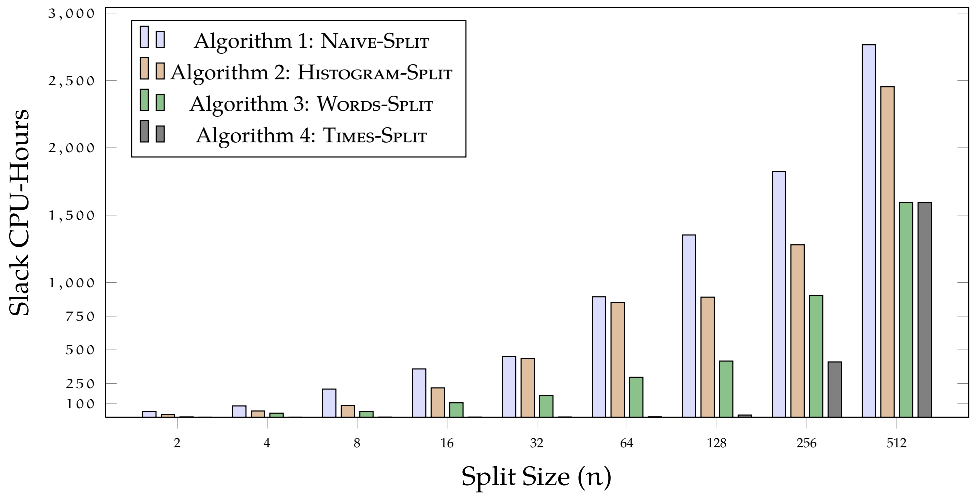
Split Size	Algorithm 1 Naive-Split		Algorithm 2 Histogram-Split		Algorithm 3 Words-Split		Algorithm 4 Times-Split	
(n)	Min	Max	Min	Max	Min	Max	Min	Max
2	365.7	408.0	376.3	397.5	386.1	387.6	*386.9*	***386.9***
4	176.1	214.4	186.8	205.0	184.6	200.9	*193.4*	***193.4***
8	84.6	122.8	84.8	107.6	88.4	101.9	*96.7*	***96.7***
16	40.8	70.8	40.5	62.0	45.3	55.0	*48.4*	***48.4***
32	19.2	38.3	18.8	37.8	20.5	29.2	*24.2*	***24.2***
64	9.2	26.1	9.2	25.4	9.4	16.7	*12.1*	***12.1***
128	2.7	16.6	4.1	13.0	3.7	9.3	5.9	**6.2**
256	0.7	10.2	1.3	8.0	1.4	6.6	2.9	**4.6**
512	0.0	6.9	0.3	6.3	0.6	**4.6**	1.1	**4.6**

(b) Decoding times in **hours** for decoder configured using a syntactic language model (Schwartz et al., 2011) addition to a 5-gram language model.

Figure 4: Decoding times for the fastest (min) and slowest (max) decoding jobs when a translation task is split into n decoding jobs. **Bold** indicates fastest max time at that split. *Italics* indicate balanced task times, corresponding to zero slack time (see Figure 5).



(a) Slack **CPU-Seconds** for decoder configured using a 5-gram language model.



(b) Slack **CPU-Hours** for decoder configured using a syntactic language model (Schwartz et al., 2011) in addition to a 5-gram language model.

Figure 5: Cumulative slack CPU time for n processing cores when processing a parallel translation task split into n jobs using various splitting algorithms. Slack CPU time is caused when some jobs finish before others. Zero slack time indicates conditions where all jobs complete simultaneously.

Algorithm 4 Split input text into n parts to balance the estimated translation time of all parts.

```
function TIMES-SPLIT($n$,input,estimate)
    for $i \leftarrow 0 \ldots ($ input.length $-1)$ do
        sentence[$i$].time $\leftarrow$ estimate[$i$]
        sentence[$i$].index $\leftarrow i$
    end for
    SORT(sentence) {|$x, y$| $y$.time $\Leftrightarrow$ $x$.time}
                                    ▷ Sort sentences by time, in reverse order
    for $i \leftarrow 0 \ldots ($ input.length $-1)$ do
        $p \leftarrow$ LEAST(times)
                                    ▷ Find partition with least time
        output[$p$].append(input[sentence[$i$].index])
        times[$p$] $\leftarrow$ times[$p$] $+$ sentence[$i$].time
    end for
    return output
end function
```

5. Conclusion

While statistical translation models could, in theory, condition on previously translated sentences, in practice virtually no widely used models do so. Translation is therefore embarrassingly parallel — a document to be translated can be split into n parts, with each part translated independently on a different computational node.

While such splitting is commonly performed, a suboptimal naive splitting technique (Algorithm 1) is used by all translation software of which we are aware. In this work we have presented three more effective corpus splitting algorithms (Algorithms 2, 3 and 4), enabling substantial speed-ups in parallel decoding time at virtually no additional cost.

We observe that while Algorithm 2 fails to improve over Algorithm 1 for a standard Moses configuration for small values of n, for values of $n > 8$, and for all values of n using the slow syntactic language model, Algorithm 2 represents a clear improvement. Results for Algorithm 3 show further speedups over Algorithms 1 and 2 in most configurations. Algorithm 4 is the clear winner, nearly eliminating slack time in many cases.

While the most effective algorithm (Algorithm 4) requires per-sentence decode times from previous decodes, in most realistic settings, Algorithm 3 provides much of the benefit of Algorithm 4 in terms of decreased computational slack time while requiring little changes to existing decoding scripts which use the naive Algorithm 1.

Implementations in Ruby and Java of all four splitting algorithms are provided at `http://github.com/dowobeha/balance-corpus`.

Bibliography

Cheng, T.C.E. and C.C.S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1999.

De, Prabuddha and Thomas E. Morton. Scheduling to minimum makespan on unequal parallel processors. *Decision Sciences*, 11(4):586–602, October 1980.

Graham, Ron L. Bounds on certain multiprocessing timing anomalies. *The Bell Systems Technical Journal*, 45(9):1563–1581, November 1966.

Graham, Ron L. Bounds on certain multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.

Hu, T.C. Parallel sequencing and assembly line problems. *Operations Research*, 9(6):841–848, 1961.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007.

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

Och, Franz and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Philadelphia, Pennsylvania, July 2002.

Panwalkar, S.S. and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1): 45–61, 1977.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Toulouse, France, 2001.

Schwartz, Lane, Chris Callison-Burch, William Schuler, and Stephen Wu. Incremental syntactic language models for phrase-based translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Techologies*, pages 620–631, Portland, Oregon, USA, June 2011.

Address for correspondence:
Lane Schwartz
`Lane.Schwartz@wpafb.af.mil`
2255 H St, Dayton, OH 45433, USA

Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force. Cleared for public release on 19 Jan 2011. Originator reference number RH-11-107802. Case number 88ABW-2011-0185. Thanks to Grant Erdmann and Tim Anderson, and to the anonymous reviewers.

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 98 OCTOBER 2012 121–131

DELiC4MT: A Tool for Diagnostic MT Evaluation over User-defined Linguistic Phenomena

Antonio Toral, Sudip Kumar Naskar, Federico Gaspari, Declan Groves

School of Computing, Dublin City University

Abstract

This paper demonstrates DELiC4MT, a piece of software that allows the user to perform diagnostic evaluation of machine translation systems over linguistic checkpoints, i.e., source-language lexical elements and grammatical constructions specified by the user. Our integrated tool builds upon best practices, software components and formats developed under different projects and initiatives, focusing on enabling easy adaptation to any language pair and linguistic phenomenon. We provide a description of the different modules that make up the tool, introduce a web demo and present a step-by-step case study of how it can be applied to a specific language pair and linguistic phenomenon.

1. Introduction

DELiC4MT[1][2] is an open-source tool for diagnostic evaluation of Machine Translation (MT) which has been developed as part of the FP7 CoSyne project.[3] In contrast to automatic MT evaluation metrics, which are only effective at carrying out overall evaluations of MT systems (either at sentence or document level), this tool allows the evaluation of MT systems over linguistic phenomena specified by the user.

Most of the software tools for MT evaluation developed during the last decade belong to the category of automatic metrics. These are programs that, given the

[1]`http://www.computing.dcu.ie/~atoral/delic4mt/`

[2]`https://github.com/antot/DELiC4MT`

[3]`http://cosyne.eu`

Corresponding author: `atoral@computing.dcu.ie`

output of an MT system and reference translation(s), apply different (primarily *n*-gram based) algorithms that provide a score by comparing the output of the system with the reference(s). Such automatic metrics include BLEU (Papineni et al., 2002), TER (Snover et al., 2006), METEOR (Banerjee and Lavie, 2005), etc. Asiya (Giménez and Màrquez, 2010) is a toolkit that provides a common interface to a rich set of metrics, thus making it easier for MT users to make use of these metrics. Asiya also incorporates schemes for metric combination.

Woodpecker (Zhou et al., 2008) is a piece of software that can evaluate MT systems over specific linguistic phenomena, also known as linguistic checkpoints (linguistically-motivated units e.g. an ambiguous word, a noun phrase, etc.), providing more fine-grained linguistically-motivated evaluation than the aforementioned 'traditional' metrics. The tool presented in the current paper builds on the paradigm introduced by Woodpecker and overcomes two of its limitations: (i) its implementation is language-independent (while Woodpecker had language-dependent data for English–Chinese hardcoded), and (ii) the license of the tool presented here allows anyone to work on it and release modifications, while conversely, Woodpecker's license, MSR-LA,[4] is quite restrictive in this regard. Moreover, the current tool provides additional functionalities such as checkpoint filtering based on PoS tags and statistical significance testing.

Evaluations of different MT systems for a range of linguistic checkpoints have been carried out for English–Chinese (Zhou et al., 2008), Italian–English, German–English and Dutch–English (Naskar et al., 2011).

The rest of the paper is structured as follows. Section 2 introduces the software architecture of the tool. This is followed by an illustrative case study in which the software is applied to a specific language pair and linguistic checkpoint. Finally we draw some conclusions and outline directions for future work.

2. Architecture

The aim of DELiC4MT is to provide the required functionality to perform diagnostic evaluation on a set of linguistic checkpoints. This is done by extracting checkpoint instances from text using PoS tagging (applied only to the source and reference translations) and word alignment and then evaluating these instances. The main focus during its development has been to allow for easy adaptation to any language pair and linguistic phenomenon. In fact, it has been applied successfully to evaluate a set of MT systems over a set of language directions involving German, Italian, Dutch and English, where the set of checkpoints is different for each language (Naskar et al., 2011). The work presented in this paper extends the work previously described in (Naskar et al., 2011) by incorporating a length-based penalty to penalize longer candidate translations as in (Zhou et al., 2008) and filtering of noisy checkpoint instances.

[4]`https://research.microsoft.com/en-us/projects/pex/msr-la.txt`

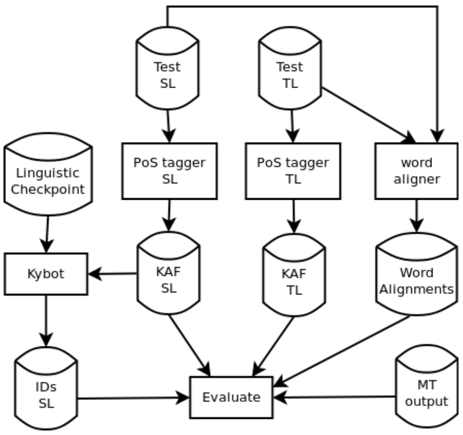


Figure 1. Architecture for linguistic checkpoints-based diagnostic evaluation of machine translation

This paper also provides additional technical details regarding the architecture and implementation of the tool.

The tool makes use of open-source software components and representation standards developed by the research community in recent years. It uses:

- State-of-the-art PoS taggers and word aligners. Treetagger[5] and GIZA++ (Och and Ney, 2003),[6] respectively, are used in the current version, although any similar tool could be used.
- KAF (Bosma et al., 2009), established in the FP7 KYOTO project,[7] for representing textual analysis. KAF is a unique format for representing all the levels of linguistic analysis based on ISO standards for each of those levels (i.e., MAF for morphology, SynAF for syntax and SemAF for semantics). Scripts to convert the output of several state-of-the-art tools are available (e.g. TreeTagger).
- Kybots (Vossen et al., 2010),[8] also developed within KYOTO, to define the linguistic phenomena to be evaluated. A Kybot profile can be thought of as a regular expression over elements and attributes in KAF documents.

Figure 1 presents the architecture of the tool. The source- and target-language sides of the gold standard (test set) are processed by PoS taggers and converted into KAF. The test set is also word aligned, and the identifiers of the aligned tokens are

[5]http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

[6] https://code.google.com/p/giza-pp/

[7]http://www.kyoto-project.eu/

[8]https://kyoto.let.vu.nl/svn/kyoto/trunk/modules/mining_module/

stored. Kybot profiles covering the different evaluation targets (linguistic checkpoints) are run on the source KAF text, and the identifiers of the matched terms are stored. Finally, the evaluation module takes as input the identifiers from the Kybot output, the KAF annotated test sets, the word alignments and the output of the MT system,[9] and calculates the performance of the MT system over the linguistic checkpoint(s) considered.

3. Case study

This section takes a closer look at the different modules of DELiC4MT by presenting a case study over a specific language pair and linguistic checkpoint. The case study demonstrates the use of the tool step-by-step.[10] A broader evaluation over a set of language pairs and checkpoints can be found in (Naskar et al., 2011). A web interface for the tool has also been developed, screenshots of which can be found at `http://www.computing.dcu.ie/~atoral/delic4mt/webdemo`. All of the scripts for running DELiC4MT are packaged within a single wrapper script (`delic4mt.sh`), facilitating ease-of-use.

3.1. Preparing the test data to be evaluated

We use the Italian–English test data of (Toral et al., 2011), consisting of source file *en-it.it* and target *en-it.en*, for illustration purposes.

The test set is PoS tagged using TreeTagger which also performs sentence-splitting and tokenisation. Since the tokens and sentences need to correspond to those in the alignment, we needed to alter TreeTagger's behaviour. A script has been developed for that reason (`treetagger_preserving_tokens_and_lines.pl`); it receives as input the text tokenised and applies TreeTagger to each sentence. The output of TreeTagger is post-processed overwriting any end-of-sentence (`SENT`) PoS tag by `OTHER`. Finally the tag of the last token of the sentence is overwritten to `SENT`. The output of this procedure is processed by a script that converts it to KAF (`treetagger2kaf.pl`). The following pipeline PoS tags the test set:

```
cat en-it.it | tokenizer.perl | treetagger_preserving_tokens_and_lines.pl \
italian | treetagger2kaf.pl -ri > en-it.it.kaf

cat en-it.en | tokenizer.perl | treetagger_preserving_tokens_and_lines.pl \
english | treetagger2kaf.pl -ri > en-it.en.kaf
```

The following is a sample of the KAF files produced for the Italian–English sentence pair 62 ("[...] la carne americana [...]", "[...] American meat [...]") [11]:

[9]The MT output is taken as is; no processing is required as for the test set.

[10]A more technical tutorial is included with the software. The tool also provides a wrapper that encapsulates all the functionalities in a single command.

[11]"carne" is the Italian word for "meat", and "americana" is the translation of "American" (inflected for feminine singular, to agree grammatically with "carne"). Note that in Italian the attributive adjective

```
<text>[...]
 <wf wid="w62_4" sent="62">la</wf>
 <wf wid="w62_5" sent="62">carne</wf>
 <wf wid="w62_6" sent="62">americana</wf>
[...]</text>
<terms>[...]
 <term tid="t62_5" type="open" lemma="carne" pos="NOM">
  <span><target id="w62_5"/></span>
 </term>
 <term tid="t62_6" type="open" lemma="americano" pos="ADJ">
  <span><target id="w62_6"/></span>
 </term>
[...]</terms>

<text>[...]
 <wf wid="w62_3" sent="62">American</wf>
 <wf wid="w62_4" sent="62" para="1">meat</wf>
[...]</text>
<terms>[...]
 <term tid="t62_3" type="open" lemma="American" pos="JJ">
  <span><target id="w62_3"/></span>
 </term>
 <term tid="t62_4" type="open" lemma="meat" pos="NN">
  <span><target id="w62_4"/></span>
 </term>
[...]</terms>
```

The test set needs to be aligned at word level so that target equivalents of the source-language checkpoints can be identified. This is done by appending them to a bigger parallel corpus, e.g., Europarl,[12] in order to help ensure accurate word alignments and avoid data sparseness. Of course, as with all alignment approaches, using in-domain parallel data, if available, would help ensure the accuracy of the word alignments, but for our work we make use of freely available data resources. The additional checkpoint filtering step (cf. Section 3.3) helps to circumvent any potential noisy alignments. The text is preprocessed with the Europarl tokeniser; then GIZA++ is applied, returning word alignments between the source and target sentences that make up the test set. The word alignments for the Italian–English sentence pair presented earlier are shown below:[13]

```
... 4-3 5-2 ...
```

3.2. Creating a linguistic checkpoint

Kybots are used to define linguistic phenomena (and extract their instances) that are to be evaluated. A Kybot profile specifies which information to extract from the KAF documents. For example the Kybot profile presented below extracts under the element "event" the term identifiers of those nouns that are immediately followed by

normally (though not necessarily) follows the noun which it modifies, whereas in English the standard order is to have the adjective first, followed by the noun.

[12] `http://www.statmt.org/europarl/`

[13] Note that identifiers in the alignment start from 0 while in the KAF files they start from 1.

an adjective in the Italian side of the test set. Target equivalents of the tokens identified by the Kybots in the source are obtained using the word alignments.

```
<Kybot id="kybot_n_a_it">
  <variables>
    <var name="X" type="term" pos="NOM*"/>
    <var name="Y" type="term" pos="ADJ*"/>
  </variables>
  <relations>
    <root span="X"/>
    <rel span="Y" pivot="X" direction="following" immediate="true"/>
  </relations>
  <events>
    <event eid="" target="$X/@tid" lemma="$X/@lemma" pos="$X/@pos"/>
    <role rid="" event="" target="$Y/@tid" lemma="$Y/@lemma" pos="$Y/@pos" rtype="follows"/>
  </events>
</Kybot>
```

For the sake of clarity, the case study presents a rather simple checkpoint. However, the expressive power of the Kybot engine allows the representation of more complex linguistic phenomena. As an example, we used a checkpoint for Italian→English that extracts sequences consisting of a noun followed by the preposition "di" followed by another noun, as there are a range of possible translations of this construction into English that are acceptable (at least in principle), e.g. keeping the preposition "of" between the two English nouns, using the genitive/possessive, or simply juxtaposing the two nouns in the target language. In order to define this checkpoint, one needs to select terms according to different fields, i.e., the first and third according to the PoS tag while the second according to both the lemma and the PoS tag.

The following commands load the Italian test file in KAF and the Kybot profile:

```
doc_load.pl --container-name docs_it en-it.it.kaf
kybot_load.pl --container-name kybots_it kybot_n_a_it.xml
```

Then the Kybot profile can be applied on the KAF document, and the matching terms are output:

```
kybot_run.pl --dry-run --profile-from-db --container-name docs_it --kybot-container-name \
kybots_it kybot_n_a_it.xml > out_n_a_it.xml
```

The following sample of the output shows the term "carne americana", terms 5 and 6 extracted from sentence 62, as it is a noun adjective sequence:

```
<kybotOut>
  <doc shortname="en-it.it.kaf">
   [...]
   <event eid="e66" target="t62_5" lemma="carne" pos="NOM"/>
   <role rid="r66" event="e66" target="t62_6" lemma="americano" pos="ADJ" rtype="follows"/>
   [...]
  </doc>
</kybotOut>
```

3.3. Filtering checkpoint instances

Optionally the tool can filter checkpoints based on corresponding PoS tags (recommended as it alleviates word alignment errors). This is done by establishing constraints based on PoS tags mappings between checkpoints extracted and the equivalent tokens in the target language (e.g., `NOM*=N*`, indicating that the equivalent token in the target language of a token in the source with PoS tag `NOM*` must have the PoS tag `N*`). If a constraint is not fulfilled the corresponding instance of the checkpoint is dropped. Consider the following two constraints for Italian→English:

```
NOM* = N*
ADJ* = JJ*
```

The following sample of the Kybot output shows a term made up of tokens 24 (index 23 in word alignment) and 25 (index 24 in word alignment) from sentence 1:

```
<event eid="e1" target="t1_24" lemma="sinodo" pos="NOM"/>
<role rid="r1" event="e1" target="t1_25" lemma="patriarcale" pos="ADJ" rtype="follows"/>
```

Consider then the word alignments of the sentence.

```
... 23-22 22-23 21-24 26-24 24-25 ...
```

The tokens of the checkpoint instance (source tokens 23 and 24) get aligned to target tokens 22 and 25 (23 and 26 in the KAF file). Now, let us check these tokens in the target language.

```
[...]
<wf wid="w1_23" sent="1">of</wf>
<wf wid="w1_24" sent="1">the</wf>
<wf wid="w1_25" sent="1">Maronite</wf>
<wf wid="w1_26" sent="1">Patriarchal</wf>
<wf wid="w1_27" sent="1">Synod</wf>
[...]
<term tid="t1_23" type="open" lemma="of" pos="IN">
<span><target id="w1_23"/></span>
</term>
<term tid="t1_26" type="open" lemma="Patriarchal" pos="NP">
<span><target id="w1_26"/></span>
</term>
[...]
```

Thus, "sinodo patriarcale" is wrongly aligned to "of Patriarchal" (the right alignment is "Patriarchal Synod"). In PoS terms, `NOM ADJ` gets aligned to `IN NP`. The constraints are checked and they are not fulfilled in this case, as the PoS correspondence for "sinodo→of" (`ADJ=IN`) does not match the constraint `ADJ=JJ`. Thus, this instance of the checkpoint is filtered out.

3.4. Evaluating MT systems on the linguistic checkpoint

The performance of a MT system over a linguistic checkpoint is calculated by using an *n*-gram based evaluation metric. We split each system-generated translation and reference for a checkpoint into a set of *n*-grams, compute the number of matching *n*-grams and sum up the gains over all the *n*-grams as the score for this checkpoint. If the reference of the checkpoint is not consecutive, we use a wildcard character ("*") which can be matched by any word sequence. Below are some examples for the Italian→English language direction to demonstrate the *n*-gram matching.

When we calculate the recall of a set of checkpoints C, the references r of all checkpoints c in C (c can be a single checkpoint, a category, or a category group) are merged

Consecutive checkpoint:

Checkpoint:	Le proteste per la carne americana
Reference:	Protests over American meat
Candidate:	The protests for the American meat
Matched n-grams:	protests, American, meat, American meat

Non-consecutive checkpoint:

Checkpoint:	Le proteste * carne [*]
Reference:	Protests * meat
Candidate:	The protests for the American meat
Matched n-grams:	protests, meat, protests * meat

into one reference set R and the recall of C is obtained on R using equation 1.

$$R(C) = \frac{\sum_{r \in R} \sum_{ngram \in r} match(ngram)}{\sum_{r \in R} \sum_{ngram \in r} count(ngram)} \tag{1}$$

Since *n*-grams appearing in the target equivalents of instances of a linguistic phenomenon are searched for in the candidate translations, longer candidate translations have a better chance of returning higher scores. So we have implemented a length-based penalty to penalize longer candidate translations. We set the length-based penalty as in 2:

$$penalty = \begin{cases} \frac{length(R)}{length(C)} & \text{if length(C) >length(R)} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

where $length(C)$ is the average candidate sentence length and $length(R)$ is the average reference sentence length. Thus systems producing longer candidate sentences are penalized. The final score is calculated as in 3:

$$Score(C) = R(C) * penalty \tag{3}$$

The evaluation module is run in the following way:

```
java -jar delic4mt.jar -alg it-en.alignment -sl_kaf en-it.it.kaf -tl_kaf en-it.en.kaf \
 -lc out_n_a_it.xml -run mt_output.iten > mt_output.iten.n
```

The five parameters are: *alg* - word alignments for the gold standard (test set), against which the evaluation is performed; *sl_kaf* - source side of the test set in KAF; *tl_kaf* - target side of the test set in KAF; *lc* - Kybot output (i.e., instances of the checkpoint); and *run* - the output of a MT system, which is to be evaluated.

Below there is sample output which shows the matching of a translation hypothesis with the reference for the checkpoint instance "carne americana".

```
Sen_id: 62
Linguistic checkpoint identified on the Source: carne americana
Target equivalent (Reference): American meat
```

	Google	Bing	Systran
Score	0.6350	0.5537	0.4806

Table 1. Scores for the MT systems on the linguistic checkpoint `N ADJ`

```
Checking for n-gram matches for checkpoint instance: 65
Ref: American meat
Hypo: The protests for the American meat , [...]

n_gram matches : American, meat, American meat
# of n-grams in reference = 3
# of matching n-grams in hypothesis 62 = 3
```

The evaluation module finally sums up the number of matching *n*-grams (across the whole testset) for the linguistic checkpoint, and divides it by the total number of (checkpoint) *n*-grams in the reference set. The scores obtained by three online MT systems (Google Translate,[14] Microsoft Bing,[15] and Systran[16]) when evaluated over the example linguistic checkpoint (`N ADJ`) are shown in Table 1. Google obtains the highest score, 8.13 points higher than Bing, which in its turn is 7.13 points higher than Systran.

3.5. Statistical Significance Tests

Finally, for each pair of systems we can check whether the difference is statistically significant. A script included provides this functionality using paired bootstrapping resampling (Koehn, 2004). Let us check if the differences between Google's and Bing's outputs are significant:

```
lingcheckp_sig.pl google.iten.n bing.iten.n
Num results: 1204, times iterations: 5, num elements per iteration: 0.3
Randomised bootstrapping 6020 iterations with 361 elements
System a better than b in 6020 iterations out of 6020, i.e. 100%
```

There are 1,204 instances, we run 1,024 · 5 iterations with 30% of the instances in each iteration (randomly selected). This means running 6,020 iterations with 361 instances each. For all of the iterations the score of system a is higher than that of system b, thus we can say that the difference is statistically significant for $p = 0.01$.

4. Conclusions and Future Work

This paper has presented DELiC4MT, a tool for evaluating MT systems over user-specified linguistic phenomena. The tool makes extensive use of already available open-source software and standards and is easily adaptable to new languages and

[14]`http://translate.google.com`

[15]`http://www.microsofttranslator.com`

[16]`http://www.systran.co.uk`

linguistic phenomena. We have presented a case study which illustrates how the tool can be adapted to evaluate a specific linguistic phenomenon of a given language pair.

Regarding future work, we envisage the following tasks: (i) use alternative aligners and alignment heuristics to investigate if we can extract accurate alignments without the need for a significant amount of additional parallel data, (ii) compare the correlation of the diagnostic evaluation metric against that of other existing automatic evaluation metrics as well as against human judgements, and (iii) introduce a precision-based component into the diagnostic evaluation metric.

Acknowledgements

This work has been funded in part by the European Commission through the CoSyne project (FP7-ICT-4-248531) and Science Foundation Ireland (Grant No. 07/CE /I1142) as part of the Centre for Next Generation Localisation (www.cngl.ie) at Dublin City University.

Bibliography

Banerjee, Satanjeev and Alon Lavie. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, Proceedings of the ACL-05 Workshop*, pages 65–72, University of Michigan, Ann Arbor, Michigan, USA, 2005.

Bosma, W. E., Piek Vossen, Aitor Soroa, German Rigau, Maurizio Tesconi, Andrea Marchetti, Monica Monachini, and Carlo Aliprandi. Kaf: a generic semantic annotation format. In *Proceedings of the GL2009 Workshop on Semantic Annotation*, Sept. 2009.

Giménez, Jesús and Lluís Màrquez. Asiya: An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation. *The Prague Bulletin of Mathematical Linguistics*, (94):77–86, 2010.

Koehn, Philipp. Statistical significance tests for machine translation evaluation. In Lin, Dekang and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 388–395, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Naskar, Sudip Kumar, Antonio Toral, Federico Gaspari, and Andy Way. A framework for diagnostic evaluation of mt based on linguistic checkpoints. In *Proceedings of the 13th Machine Translation Summit*, pages 529–536, Xiamen, China, September 2011.

Och, Franz Josef and Hermann Ney. A systematic comparison of various statistical alignment models. *Comput. Linguist.*, 29:19–51, March 2003. ISSN 0891-2017. doi: http://dx.doi.org/10.1162/089120103321337421.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1073083.1073135.

Snover, Mathew, Bonnie Dorr, Richard Schwartz, John Makhoul, and Linnea Micciula. A Study of Translation Error Rate with Targeted Human Annotation. In *AMTA 2006: Proceedings of* *the 7th Conference of the Association for Machine Translation in the Americas: Visions for the Future of Machine Translation*, pages 223–231, Cambridge, Massachusetts, USA, 2006.

Toral, Antonio, Federico Gaspari, Sudip Kumar Naskar, and Andy Way. A comparative evaluation of research vs. online mt systems. In Forcada, Mikel L., Heidi Depraetere, and Vincent Vandeghinste, editors, *Proceedings of the 15th Annual Conference of the European Associtation for Machine Translation*, pages 13–20, Leuven, Belgium, 2011. European Association for Machine Translation.

Vossen, Piek, German Rigau, Eneko Agirre, Aitor Soroa, Monica Monachini, and Roberto Bartolini. Kyoto: an open platform for mining facts. In *Proceedings of the 6th Workshop on Ontologies and Lexical Resources* , pages 1–10, Beijing, China, August 2010. Coling 2010 Organizing Committee.

Zhou, Ming, Bo Wang, Shujie Liu, Mu Li, Dongdong Zhang, and Tiejun Zhao. Diagnostic evaluation of machine translation systems using automatically constructed linguistic checkpoints. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 1121–1128, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. ISBN 978-1-905593-44-6.

Address for correspondence:
Antonio Toral
`atoral@computing.dcu.ie`
School of Computing, Dublin City University, Dublin 9, Ireland

The Prague Bulletin of Mathematical Linguistics
NUMBER 98 OCTOBER 2012 133–142

Extending Hiero Decoding in Moses with Cube Growing

Wenduan Xu[a], Philipp Koehn[b]

[a] Computer Laboratory, University of Cambridge
[b] School of Informatics, University of Edinburgh

Abstract

Hierarchical phrase-based (Hiero) models have richer expressiveness than phrase-based models and have shown promising translation quality gains for many language pairs whose syntactic divergences, such as reordering, could be better captured. However, their expressiveness comes at a high computational cost in decoding, which is induced by huge dynamic programs associated with language model integrated decoding, where the search space is lexically exploded and exact search often becomes intractable. Cube pruning and growing are two approximate search algorithms to make decoding much more efficient. In this article, we describe an extension to the Hiero decoder of the Moses toolkit by providing cube growing as an alternative to cube pruning, with an additional parameter similar to Jane's cube growing implementation that is not present in the original one. We also report experimental results on a full-scale NIST MT08 Chinese-English translation task.

1. Introduction

Cube pruning for machine translation (MT) decoding performs lazy computation along multi-hyperedges in parallel. However, it still computes a full *k*-best list for each node in the hypergraph. Based on this observation, Huang and Chiang (2007) further propose a lazy variant of cube pruning, known as *cube growing*, derived from *k*-best parsing in Huang and Chiang (2005), which turns the *k*-best selection problem into a depth-first, top-down recursive *k*-best generation procedure, and only generates as many hypotheses as needed at each hypergraph node to obtain the k^{th} best hypothesis of the root node.

In Hiero decoding, cube growing is a two-pass procedure. In the first pass translation model only monotonic (`-LM`) decoding, a translation hypergraph is generated.

Corresponding author: pkoehn@inf.ed.ac.uk
Cite as: Wenduan Xu, Philipp Koehn. Extending Hiero Decoding in Moses with Cube Growing. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 133–142. doi: 10.2478/v10108-012-0015-8.

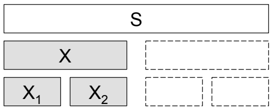


Figure 1. A toy derivation in a hypergraph. The hyperedge associated with the SCFG rule $X \rightarrow X_1 X_2$ *is shaded.*

This pass could be treated as an initial bottom-up rule look-up only phrase, even though more housekeeping work is carried out in this first pass as preparation for the second-pass top-down main cube growing procedure. The main cube growing is then applied to the resulting hypergraph to generate the *k*-best hypotheses of the goal node (the *k*-best translations of a given complete sentence) in a top-down manner.

The original cube growing algorithm by Huang and Chiang (2007) is applied to a tree-to-string translation model, and the authors have left its extension to the Hiero model as part of their future work. We adapted the cube growing algorithm to the Hiero model and implemented it as an alternative search algorithm in addition to cube pruning in Moses (Koehn et al., 2007). Moreover, inspired by the cube growing implementation by Vilar et al. (2010)[1], we have introduced an additional parameter not present in the original cube growing algorithm to boost its search and translation quality.

2. Cube Growing for Hiero Decoding

We first provide an overview of the main structure of the cube growing algorithm, then follow this up with an example concentrating on a critical detail of it. We depict one possible derivation of the goal node of a hypergraph in Figure 1. Suppose we are interested in finding the first-best hypothesis of S (for the sake of discussion, we ignore the immediate right child of S, and concentrate only on the immediate left child node X). First, the main cube growing procedure is called on the S node, it will then request the first-best hypothesis of the X node. However, this required first-best hypothesis has not been generated yet, and the X node will call the main cube growing procedure on itself, which causes two further recursive calls of the main cube growing procedure on the two leaf nodes X_1 and X_2 (assuming it is the first time we visit these two leaf nodes, the first-best hypotheses of them are unknown either). Using a subprocedure of cube growing, the two leaf nodes will return the recursive calls to node X with their respective first-best hypothesis (assume for now the two leaf-nodes used a black-box subprocedure to return their first-best hypotheses to X). Upon receiving the first-best hypotheses of both X_1 and X_2, node X will use the same black-box subprocedure to

[1]To the best of our knowledge, Jane was the only open-source decoder which implements cube growing.

0.6	0.4	0.3
0.1	0.5	0.6
0.7	0.5	0.2

Figure 2. Language model costs for all hypotheses associated with a hyperedge, lower costs are better.

determine if the candidate first-best hypothesis generated from the given two first-best hypotheses of X_1 and X_2 is good enough to be passed up to node S. If not, more recursive calls would be made to both of its child nodes until it is confident to pass a hypothesis as its first-best hypothesis to S. Similarly, node S would use the same black-box subprocedure to determine its first-best hypothesis.

We now turn to the critical subprocedure and concentrating on cube growing along one hyperedge. In short, cube growing uses an estimated minimal language model cost, termed the language model intersection cost heuristic to determine whether a hypothesis is good enough to be enumerated into the final *k*-best list of a hyperedge. To define the language model intersection cost heuristic, assume for the moment that all language model intersection costs for a hyperedge are known, as depicted in each cell of Figure 2, and the language model heuristic for this hyperedge is just $\min\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\} = 0.1$. In general, with $\mathrm{LM}(\mathbf{h}_e)$ denoting the language model cost of a hypothesis $\mathbf{h}$ from the hyperedge e, the language model intersection cost heuristic for this hyperedge is

$$\tau = \min_{\mathbf{h}_e \in \mathbf{H}} \mathrm{LM}(\mathbf{h}_e),$$

where $\mathbf{H}$ is the set of all possible hypotheses of this hyperedge. We note that this language model heuristic is a lower bound on language model intersection costs along this hyperedge, and we could estimate the total cost of any hypothesis $\mathbf{h}_e$ generated from this hyperedge as $\beta = \lambda(\mathbf{h}_e) + \tau$, where $\lambda(\mathbf{h}_e)$ is the total cost disregarding the language model cost. Also, β is always an underestimate if τ is the true minimal language model intersection cost along the hyperedge e. Furthermore, we denote the *true* total cost of a hypothesis $\mathbf{h}_e$ as $\alpha = \lambda(\mathbf{h}_e) + \mathrm{LM}(\mathbf{h}_e)$.

Return to our example and focus on cube growing along the hyperedge $X \rightarrow X_1X_2$ as illustrated in Figure 3, assuming we know in advance that $\tau = 0.1$ for this hyperedge. In Figure 3(a) (top cube) the first recursive calls for X_1 and X_2 have returned, yielding a total cost of 1.0 for both of the first-best hypotheses of X_1 and X_2. A candidate first-best item of X is then generated, as shown in the top cube of Figure 3(a), its α cost is $1.0 + 1.0 + 0.7 = 2.7$, with 0.7 as the language model cost. This item is first put into a priority queue (*PriorityQueue*), and then is immediately popped out

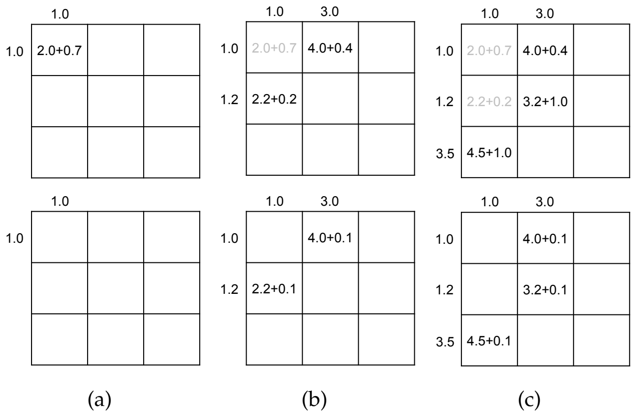


Figure 3. Example illustrating cube growing along one hyperedge. The two axes of all the "cubes" correspond to hypotheses for X_1 and X_2 respectively. On the top row, black numbers indicate α costs of hypotheses in the PriorityQueue, and grey numbers indicate hypotheses in the PriorityQueue-temp. For each cube on the top row, the corresponding cube on the bottom row represents another priority queue which contains the same hypotheses as the cube above it, but with β costs as the priorities.

of the queue as it is the only item in the queue and pushed into a buffer priority queue (*PriorityQueue-temp*) for further comparison with more items (as we are not certain this item is the true first-best item of X due to non-monotonicity). More recursive calls cause more candidate items being generated at the node X and the top cube of Figure 3(b) shows two more candidate items with α costs $2.2 + 0.2 = 2.4$ and $4.0 + 0.4 = 4.4$ and these two items are again put into the *PriorityQueue*. Moreover, as we are given that $\tau = 0.1$ for this hyperedge, we have the estimated total costs (the β costs) for these two items as $2.2 + 0.1 = 2.3$ and $4.0 + 0.1 = 4.1$. For each cube on the top row, we record β costs of hypotheses in the *PriorityQueue* into a corresponding monotonic grid on the bottom row of Figure 3 (in a practical implementation, we could implement this monotonic grid as another priority queue).

Now *PriorityQueue-temp* contains a single item $\mathbf{h}_e\left(\mathbf{1}\right)$ with an α cost of 2.7 depicted in grey (top cube of Figure 3(b)) and *PriorityQueue* contains two items with $\alpha(\mathbf{h}_e\left(2,1\right)) = 2.4$, $\alpha(\mathbf{h}_e\left(1,2\right)) = 4.4$, depicted in black (top cube of Figure 3(b)) and $\beta(\mathbf{h}_e\left(2,1\right)) = 2.3$, $\beta(\mathbf{h}_e\left(1,2\right)) = 4.1$ (bottom cube of Figure 3(b)). As in Huang and Chiang (2007), we define

$$\text{bound} = \min_{\mathbf{h}_e \in \text{PriorityQueue}} \beta\left(\mathbf{h}_e\right),$$

and in this example, $\mathtt{bound} = \mathtt{min}\{\beta(h_e\,(2{,}1)), \beta(h_e\,(1{,}2))\} = \mathtt{min}\{2.3, 4.1\} = 2.3$ with β costs as recorded in the bottom cube of Figure 3(b). Because $\alpha\,(h_e\,(1)) > \mathtt{bound}$, i.e., $2.7 > 2.3$, assuming lower cost is better, the candidate first-best item cannot be popped out of *PriorityQueue-temp* as the first-best hypothesis of this hyperedge (because $\mathtt{bound}$ tells us the α costs of any future items along this hyperedge could be anything greater than 2.3, hence $h_e\,(1)$ with an α cost of 2.7 may not be the true first-best, assuming lower cost is better). As such, we continue popping the minimal α cost hypothesis $h_e\,(2{,}1)$ from the *PriorityQueue* and put it into the *PriorityQueue-temp*, as shown in the top cube of Figure 3(c). More recursive calls give us the neighbours of this popped out hypothesis, with α costs $4.5 + 1.0 = 5.5$ and $3.2 + 1.0 = 4.2$ respectively. These two new neighbours are put into the *PriorityQueue* and their β costs are recorded as in the bottom cube of Figure 3(c). Now there are three items in the *PriorityQueue* (indicated by the three black numbers in the top cube of Figure 3(c)). Taking the minimal β costs of the three items (as depicted in the bottom cube of Figure 3(c)) in the *PriorityQueue*, we get $\mathtt{bound} = \mathtt{min}\{4.6, 3.3, 4.1\} = 3.3$. This time, α costs of both of the items in the *PriorityQueue-temp* are less than *bound*, and therefore could be enumerated into the final *k*-best lists, with $h_e\,(2{,}1)$ as the first-best hypothesis with α cost 2.4 and $h_e\,(1)$ as the second-best hypothesis with α cost 2.7. This procedure continues along this hyperedge and eventually generates all the *k*-best hypotheses required and in this case, the generated *k*-best items would be the true *k*-best hypotheses because 0.1 is a true lower bound of the β costs (equivalently, underestimate of α costs) along this hyperedge.

3. Language Model Cost Heuristic

In the previous section, we have explained the overall structure and described the details of the cube growing algorithm along a single hyperedge. The main recipe for cube growing along one hyperedge is to use the lower bound of the language model costs of all the hypotheses from that hyperedge to guide the search (i.e., we use the lower bound to determine if a hypothesis could be popped out of *PriorityQueue-temp* and enumerated to the final *k*-best list). For a single hyperedge, if the bound is a true lower bound, then there will be no search errors and an exact *k*-best list would be obtained for that hyperedge. Analogous to cube pruning, the version of cube growing used in MT decoding is a multi-hyperedge version, and thus we need to establish whether the lower bound is still valid if we apply the above procedure to the multi-hyperedge case where the *PriorityQueue* contains hypotheses from multiple hyperedges of one consequent node in a hypergraph. Fortunately, the validity of this lower bound across multiple hyperedges of a given consequent node is formally obtained and a complete proof is given by Huang and Chiang (2007). For completeness, we show the pseudocode of cube growing for Hiero decoding in Algorithm 1.

Algorithm 1 Cube Growing for Hiero Decoding, adapted from the algorithm in Huang and Chiang (2007)

```
1: procedure DecodeChartSpanTopDown($X, k$)
2:     if PriorityQueue uninitialised then
3:         PriorityQueue $(X) \leftarrow \emptyset$
4:         LazyCreateCube $(\text{PriorityQueue}, \mathbf{1}, e)$ foreach $e \in \text{BS}(X)$
5:         PriorityQueue-temp $\leftarrow \emptyset$
6:     while $|H_{\text{top-k}}| < k$ and $|\text{PriorityQueue-temp}(X)| + |H_{\text{top-k}}| < j$ do
7:         if $|\text{PriorityQueue}| > 0$ then
8:             $\mathbf{h}_e(\mathbf{u}) \leftarrow$ PriorityQueue.pop-min$_{\preceq}$
9:             PriorityQueue-temp.push $(\mathbf{h}_e(\mathbf{u}))$
10:            LazyCreateNeighbours $(\text{PriorityQueue}, \mathbf{h}_e(\mathbf{u}))$
11:            bound $\leftarrow \min\{\beta(\mathbf{h}_e) \mid \mathbf{h}_e \in \text{PriorityQueue}\}$
12:            GenerateHypothesis $(\text{PriorityQueue-temp}, H_{\text{top-k}}, \text{bound})$
13:    GenerateHypothesis $(\text{PriorityQueue-temp}, H_{\text{top-k}}, +\infty)$

14: procedure LazyCreateNeighbours(PriorityQueue, $\mathbf{h}_e(\mathbf{u})$)
15:     LazyCreateCube $(\text{PriorityQueue}, \mathbf{u}{+}\mathbf{b}_i, e)$ foreach $i$ in $1 \ldots |e|$

16: procedure LazyCreateCube(PriorityQueue, $\mathbf{u}$, $e$)
17:     $e$ is $X \leftarrow X_1 \ldots X_{|e|}$
18:     for $\leftarrow 1 \ldots |e|$ do
19:         DecodeSpanTopDown $(X_i, \mathbf{u}_i)$
20:         if $|H_{\text{top-k}}(X_i)| < \mathbf{u}_i$ then
21:             return
22:         PriorityQueue.push $(\mathbf{h}_e(\mathbf{u}))$

23: procedure GenerateHypothesis(PriorityQueue-temp, $H_{\text{top-k}}$, bound)
24:     while $|\text{PriorityQueue-temp}| > 0$ and $\min(\text{PriorityQueue-temp}) < \text{bound}$ do
25:         $H_{\text{top-k}}$.push (PriorityQueue-temp.pop-min)
```

4. Estimating Language Model Cost Lower Bounds

As we have briefly described before, a first initial pass of `-LM` decoding is needed for the main cube growing procedure. It is in the first pass that we generate the translation hypergraph and try to estimate the language model lower bounds for each hyperedge. In a practical implementation, it is computationally too expensive to compute the true lower bound of language model intersection costs for each hyperedge, and following Huang and Chiang (2007), we adopt a less computationally demanding approach in our implementation.

In the `-LM` pass of decoding a sentence, we first generate the `-LM` top-k best translations for the complete input sentence, these k-best translations correspond to k complete derivations in the translation hypergraph and they may share some identical hyperedges. By following backpointers of these top-k `-LM` derivations, we can calculate language model intersection costs for all the hyperedges involved in those `-LM` k-best derivations using `-LM` hypotheses. For a hyperedge shared among multiple derivations, the minimal intersection cost is then taken as the language model lower bound estimate for that hyperedge, and for a hyperedge not shared by more than one derivation, its only intersection cost will be used as the language model lower bound estimate. Moreover, a hyperedge may be found not represented in the first pass at all, in that case we then take the language model cost of the first-best hypothesis at that hyperedge as the lower bound estimate. More detailed study on the miss rate of this procedure can be found in Vilar and Ney (2009).

While this procedure is not guaranteed to cover all hyperedges that would be used in the top-down main cube growing procedure, it is expected that the resulting estimates from these top-k `-LM` derivations would be sufficient to guide the search.

5. Cube Growing Practicalities

In the original cube growing algorithm, a parameter analogous to `popLimit` in cube pruning is used to control its computational cost (the variable j as shown on line 6 of Algorithm 1). This parameter is an upper bound on the number of hypotheses allowed in *PriorityQueue-temp*. Inspired by the cube growing implementation in Jane (Vilar et al., 2010), we introduce an additional parameter which is used as a lower bound on the hypothesis count in *PriorityQueue-temp*. However, our specific implementation of the lower bound parameter is different. In our implementation, the lower bound must be surpassed each time we start to add hypotheses into *PriorityQueue-temp*, while in Jane it is only used as an one-off lower bound to accumulate hypotheses when it is the first time hypotheses are being added into a *PriorityQueue-temp*. In our initial experiments, we have found that by using this lower bound parameter, it consistently improves the search and translation quality of our cube growing implementation in Moses, and thus it is used by default in our experiment.

6. Experiment

We evaluated our implementation with a full-scale NIST MT08 Chinese-English test set, which consists of 1,357 Chinese sentences. The Hiero model is trained with part of the GALE 2008 data, which has about 50M words for each language. A sentence length limit of 80 words and a symbol limit of 5 are applied. A 5-gram language model interpolated from three separate language models trained on the English side of the parallel corpus is used.

To compare with the cube pruning baseline, we decoded the test set with 21 combinations of lower and upper bounds (cf., Section 5) for the *PriorityQueue-temp* size, with each combination having equal lower and upper bounds. The first 19 bounds range from 10 to 100 in steps of 5, the last two bounds are 150 and 250, respectively. First pass `-LM` *k*-best size for cube growing is set to 200 in all experiments. BLEU and model scores against average hypothesis count plots are shown in Figure 4, with Figure 4(c) showing an enlarged part of the model cost plot. In a nutshell, cube growing maintains much more consistent runtime requirements than the cube pruning baseline and achieves more significant speedups when the `popLimit` values of cube pruning are relatively high, while maintaining similar levels of translation and search quality.

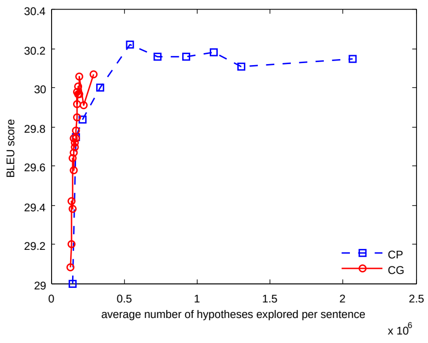
When competing with cube pruning at low `popLimit` values, cube growing has no clear advantage of speed. This is due to the fact that cube growing always requires a first bottom-up pass to generate `-LM` hypotheses and compute language model lower bound estimates. This first pass already dominates the overall runtime for cube growing with low *PriorityQueue-temp* size bounds and would have compromised the speedups gained in the top-down cube growing pass.

7. Conclusion

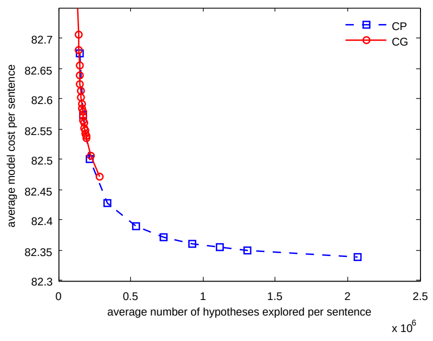
We described our implementation of the cube growing decoding algorithm originally proposed for a tree-to-string translation model (Huang and Chiang, 2007) as an alternative search algorithm for Hiero decoding in Moses and inspired by Jane (Vilar et al., 2010), a new parameter is also introduced into the original algorithm. Our experiment shows that cube growing provides a competitive alternative to cube pruning in terms of decoding speed, while maintaining the same level of translation and search quality. As future work, we would like to extend cube growing to support more syntax-based models in Moses.

Acknowledgements

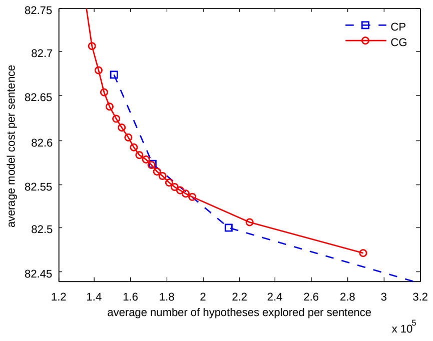
The authors would like to thank David Vilar and Huang Liang for helpful discussions and the anonymous reviewers for their constructive feedback.



(a) BLEU score vs. average hypothesis count.



(b) Average model cost (lower is better) vs. average hypothesis count.



(c) Enlarged (b). Average model cost (lower is better) vs. average hypothesis count.

Figure 4. Translation and search quality comparisons of cube pruning and cube growing.

Bibliography

Huang, Liang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

Huang, L. and D. Chiang. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, volume 45, page 144, 2007.

Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.

Vilar, D. and H. Ney. On lm heuristics for the cube growing algorithm. In *Proceedings of the Annual Conference of the European Association for Machine Translation*, pages 242–249. Association for Computational Linguistics, 2009.

Vilar, D., D. Stein, M. Huck, and H. Ney. Jane: Open source hierarchical translation, extended with reordering and lexicon models. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 262–270. Association for Computational Linguistics, 2010.

Address for correspondence:
Philipp Koehn
pkoehn@inf.ed.ac.uk
Informatics Forum
10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom

PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 98 OCTOBER 2012

INSTRUCTIONS FOR AUTHORS

Manuscripts are welcome provided that they have not yet been published elsewhere and that they bring some interesting and new insights contributing to the broad field of computational linguistics in any of its aspects, or of linguistic theory. The submitted articles may be:

- long articles with completed, wide-impact research results both theoretical and practical, and/or new formalisms for linguistic analysis and their implementation and application on linguistic data sets, or
- short or long articles that are abstracts or extracts of Master's and PhD thesis, with the most interesting and/or promising results described. Also
- short or long articles looking forward that base their views on proper and deep analysis of the current situation in various subjects within the field are invited, as well as
- short articles about current advanced research of both theoretical and applied nature, with very specific (and perhaps narrow, but well-defined) target goal in all areas of language and speech processing, to give the opportunity to junior researchers to publish as soon as possible;
- short articles that contain contraversing, polemic or otherwise unusual views, supported by 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 the original pdf files.

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`.