PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 96 OCTOBER 2011 89–98

eppex: Epochal Phrase Table Extraction for Statistical Machine Translation

Česlav Przywara, Ondřej Bojar

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Abstract

We present a tool that extracts phrase pairs from a word-aligned parallel corpus and filters them on the fly based on a user-defined frequency threshold. The bulk of phrase pairs to be scored is much reduced, making the whole phrase table construction process faster with no significant harm to the ultimate phrase table quality as measured by BLEU. Technically, our tool is an alternative to the *extract* component of the *phrase-extract* toolkit bundled with Moses SMT software and covers some of the functionality of *sigfilter*.

1. Motivation

Phrase tables in Statistical Machine Translation (SMT) systems generally take the form of a list of pairs of phrases s and t, s being the phrase from the source language and t being the phrase from the target language, along with scores that should reflect the goodness of translating s as t. The standard approach to obtain such scores is to use *maximum likelihood probability* of the phrase t given the phrase s and vice versa. The probabilities $p(s|t)$ and $p(t|s)$ are often referred to as *forward* and *reverse translation probabilities*.

To estimate $p(s|t)$ and $p(t|s)$, frequency counts $C(t, s)$, $C(s)$ and $C(t)$ are usually collected from the entire training corpus. For substantial coverage of source and target languages, such corpora are often very big so all phrase pairs and their counts cannot fit in the physical memory of the computer. To overcome this limitation, phrase table construction methods often simply dump observed phrases to local disk and sort and

Corresponding author: bojar@ufal.mff.cuni.cz
Cite as: Česlav Przywara, Ondřej Bojar. eppex: Epochal Phrase Table Extraction for Statistical Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 96, 2011, pp. 89–98.
doi: 10.2478/v10108-011-0014-1.

count them on disk. This approach allows to construct phrase tables of size limited only by the capacity of the disk. The obvious drawback of this solution is that much more time is needed to build the table.

Moses (Koehn et al., 2007), an open-source SMT toolkit with a full set of tools required for SMT system training, adheres to this concept. Both of the two main components used to construct phrase tables (*extract* to observe phrases and *scorer* to score them) treat their input as an unbounded stream of data, keeping only limited span of this stream in physical memory and using local disk for temporary storage.

It is known that phrase table quality is not strictly determined by its size. Johnson et al. (2007) presented a method for the reduction of phrase table size causing no harm to translation quality as measured by BLEU (Papineni et al., 2002). Their method employs significance testing of phrase pair co-occurrence in the parallel corpus to distinguish between valuable phrase pairs and random noise. Because significance testing of phrase pair co-occurrences is based on their frequencies, the method was designed as a post-processing filter applied to a finished phrase table. The phrase table extraction process and its runtime requirements are unaffected by this method.

In this paper, we present *eppex*, a tool designed as a drop-in alternative for *extract* component in Moses training. Like *extract*, our tool extracts phrase pairs from a word-aligned parallel corpus. Unlike *extract*, *eppex* filters out phrase pairs with frequency below a user-defined threshold. As a result, the subsequent sorting and scoring have to process a reduced set of phrase pairs, so the whole phrase table extraction pipeline requires less time to finish.

In the rest of the paper we present the implementation details and results of the experiments aiming at comparison of the standard approach, the standard approach with additional significance filtering and the epochal extraction with respect to translation quality and runtime performance.

2. Implementation

Our tool, just like the *extract* component, processes the input parallel corpus in a single pass. Our implementation reuses the code from *extract* that implements the extraction of individual phrase pairs from word-aligned parallel corpora as proposed by Och and Ney (2003). In contrast to *extract*, extracted phrase pairs are not immediately printed to a temporary storage on the disk, but instead they are fed into an algorithm that on the fly filters out low frequency items.

To carry out the filtration within manageable memory demands, we employ an algorithm for approximate frequency counting proposed by Manku and Motwani (2002). Their *Lossy Counting* algorithm expects two user-defined thresholds: *support* $s \in (0, 1)$ and *error* $\epsilon \in (0, 1)$, such that $\epsilon \ll s$. At any point of time (after being fed with N items) the algorithm can output the list of items with their approximate frequencies and guarantee the following:

- All items whose true frequency exceeds sN are output (*no false negatives*).

- No item whose true frequency is less than $(s-\epsilon)N$ is output (*few false positives*).
- Estimated frequencies are less than the true frequencies by at most ϵN.
- The space used by the algorithm is $O(\frac{1}{\epsilon}\log \epsilon N)$.

2.1. Lossy Counting Algorithm

The Lossy Counting algorithm conceptually divides the incoming stream of items into epochs of fixed size $w = \lceil \frac{1}{\epsilon} \rceil$ (thus the name *epochal extraction*). In order to deliver the frequency estimates, the algorithm maintains a data structure D consisting of triples (e, f, Δ), where e is an element from the stream, f is its estimated frequency and Δ is the maximum possible error in f. When a new item e arrives, a lookup for e in D is performed. If e is already present, its frequency f is incremented by one. Otherwise a new triple $(e, 1, T-1)$ is added to D, where T denotes the ID of current epoch (with IDs starting from 1).

At the end of each epoch (determined by $N \equiv 0 \bmod w$), the algorithm prunes off all items whose maximum true frequency is small. Formally, at the end of the epoch T, all triples satisfying the condition $f + \Delta \leqslant T$ are removed from D. When all elements in the stream have been processed, the algorithm returns all triples (e, f, Δ) where $f \geq (s-\epsilon)N$.

The idea behind the algorithm is that frequent elements show up more than once within each epoch so their frequencies are increased enough to survive the filtering.

2.2. Memory Management

To optimize the usage and access speed of the memory, we implement a couple of tricks. First, we explicitly store the vocabulary of the phrases read so far. The phrases are then represented as vectors of word indices instead of full strings. (The vocabulary is not subject to pruning at epoch boundaries for efficiency reasons.)

Second, we take advantage of the fact the vocabulary size of MT corpora usually is on the order of millions. We therefore represent words as 4-byte integers allowing to store up to 4 billions of word types. Using directly the pointer to the string representation of the word would be more expensive in a 64-bit environment.

Third, we optimize the process of memory allocation for newly created word types by using memory pools as implemented in the *Boost* library.

2.3. Usage

Eppex is implemented as C++ program and does not require any special libraries to compile and run, except for moderately recent version of *gcc*. The authors did not attempt to compile *eppex* on Windows, but the code is free of system-dependent hacks, so porting to Windows should be fairly straightforward.

Eppex input and output format is fully compatible with that of *extract*. We also keep the command line syntax very similar. The main change is that instead of the *max-*

phrase-length parameter, one has to specify the ϵ and s values for the Lossy Counting algorithm. A different pair of thresholds can be given for each phrase pair length[1] allowing for a more fine-grained pruning.

The command line syntax is:

```
eppex tgt src align extract lossy-counter [lossy-counter-2 [...]] \
  [orientation [ --model [wbe|phrase|hier]-[msd|mslr|mono] ]]
```

Every `lossy-counter` specifies the *error* and *support* for the Lossy Counting algorithm for phrases of a length within an interval. The parameter takes the form *length:error:support*, where *length* is either a number or an interval specification and *error* and *support* are two floats. For example, to keep in all phrase pairs with length 1 and prune all phrase pairs of length from 2 to 4 with $\epsilon = 2 \times 10^{-7}$ and $s = 8 \times 10^{-7}$, two lossy counters must be declared as: `1:0:0 2-4:2e-7:8e-7`.[2]

Phrases of length not covered by any `lossy-counter` are not extracted at all, effectively setting also the *max-phrase-length*.

No defaults for `lossy-counters` are provided, because reasonable settings heavily depend on the corpus. *Eppex* at its end and also a faster single-purpose tool *counter* report the total number of extracted phrases of all lengths, allowing to set the thresholds.

3. Experiments

We compared our tool against two other methods of phrase table construction that were already introduced in the beginning of the paper:

1. the *extract* component of Moses toolkit, i.e. the baseline,
2. the *sigfilter* program, which is a re-implementation of significance testing phrase table filter described by Johnson et al. (2007) and is also bundled with Moses.

The baseline scenario has no options to adjust: all phrase pairs extracted from the corpus are included in the final phrase table. When applying *sigfilter*, at least one of two options has to be set: the *cutoff threshold* or the *pruning threshold*. By setting the *cutoff threshold* to n, all but the top n phrase pairs, sorted by the forward probability $P(t|s)$, will be removed. Johnson et al. (2007) recommend the cutoff of 30. The *pruning threshold* determines the minimum level of negative-log-p-value that a particular phrase pair (s, t) has to reach under Fisher's exact test that calculates probability of observed two by two contingency table based on frequency counts $C(s)$, $C(t)$ and $C(s, t)$. A particularly interesting settings for this threshold are values $\alpha - \epsilon$ and $\alpha + \epsilon$, where $\alpha = \log(N)$ and ϵ is appropriately small positive number. The former is the largest

[1]Phrase pair length is defined as the length of the longer of the phrases.

[2]All phrases of length 2–4 are stored together in one counter. To treat them separately, the counter has to be split in three: `2:2e-7:8e-7 3:2e-7:8e-7 4:2e-7:8e-7`.

threshold that results in keeping all the 1-1-1 phrase pairs[3] in the table, whereas the latter is the smallest threshold that results in all such phrase pairs being removed.

With *eppex*, a separate lossy counting may be instantiated for each phrase pair length, allowing to filter them with different values of sN (positive) and $(s - \epsilon)N$ (negative) thresholds. Depending on the corpus, the values of *support* and *error* have to be adjusted. The effect of *sigfilter* with $\alpha \pm \epsilon$ pruning may be approximated: setting *support* s such that $sN < 1$ will preserve all of the 1-1-1 phrases (like in $\alpha - \epsilon$), while setting *support* s and *error* ϵ to satisfy $(s - \epsilon)N > 1$ will result in their complete removal (like in $\alpha + \epsilon$).

3.1. Data

We evaluate the extraction methods on English-Czech translation trained on the corpus CzEng (Bojar and Žabokrtský, 2009) with a few additions and a large Czech language model. See Mareček et al. (2011) for the exact setup of the system "cu-bojar".

The complete parallel corpus for our experiments with phrase extraction is 8.4M sentence pairs; 107.2M English and 93.2M Czech tokens.

Our tuning and testing data come from the WMT 2011 Translation Task[4].

3.2. Benchmarking

The training process of Moses SMT takes place in nine steps.[5] The steps cover the whole training pipeline including word alignment, lexical table construction, phrase table construction and more. The phrase table construction itself is done in two steps, phrase extraction and phrase scoring, which might be even further split into following substeps: (1) phrase extraction that produces direct and reverse phrase table halves (without scores yet); (2) gzipping, (3) sorting and (4) scoring of the direct table; (5) gzipping, (6) sorting and (7) scoring of the reverse table; (8) sorting of the scored reverse table; (9) consolidation of the scored direct and reverse tables; (10) gzipping of the consolidated phrase table.

The default implementation in Moses training script *train-model.perl* does not use any parallelization of the sequence (steps 2–4 and 5–7 could be run in parallel). The gzipping in steps 2 and 5 is somewhat dubious but everybody seems to use it.[6]

We benchmark phrase table construction by measuring CPU time, wall clock time and memory requirements of all the substeps. To measure the memory with a reasonable precision, we save a copy of *stat* and *status* files from */proc/[pid]/* directory of the measured process(es) every second.

[3]A phrase pair (s,t) is called 1-1-1, if $C(s) = 1$, $C(t) = 1$ and $C(s,t) = 1$.

[4]`http://www.statmt.org/wmt11/translation-task.html`

[5]`http://www.statmt.org/moses/?n=FactoredTraining.HomePage`

[6]We discovered that the option *–dont-zip* of *train-moses.perl* has been broken since it was introduced.

We run all the steps on a single machine for comparability of the results. Although the machine is a standard node in a cluster, we keep jobs of other users away by reserving all memory of the machine for our job. The machine runs the 64-bit version of Ubuntu 10.04 server edition on 2 Core4 AMD Opteron 2.8 Ghz processors with 32 GB of RAM in total. All the input and output files were read and written to a locally mounted hard disk.

4. Results

Table 1 presents all the experiments and their settings. We compare the baseline, the recommended default settings for significance filtering and two *eppex* runs: one with mild pruning that left in all shorter phrase pairs (denoted as *eppex 1-in*) and one harsher that filters out all phrase pairs with single occurrence only (*eppex 1-out*).

Name	Description
baseline	standard Moses pipeline with *extract* component
eppex 1-in	the pipeline with *eppex*: all phrase pairs of length 1–3 kept in, longer phrase pairs pruned with max. positive threshold of 8
eppex 1-out	the pipeline with *eppex*: all single-occurring phrase pairs removed, phrase pairs pruned with max. positive threshold of 8
sigfilter a-e	baseline followed by *sigfilter* with pruning threshold $\alpha - \epsilon$
sigfilter a+e	baseline followed by *sigfilter* with pruning threshold $\alpha + \epsilon$
sigfilter 30	baseline followed by *sigfilter* with cutoff threshold of 30

Table 1. List of the experiments and their settings

4.1. Translation quality

We evaluate translation quality automatically using BLEU. The complete SMT system includes also a language model (always identical) and a distortion model. For *eppex* setups, the distortion model was always built from the same (pruned) set of phrase pairs as the phrase table. In each setup separately, model weights are optimized using Moses MERT.

Table 2 presents BLEU scores obtained in the experiments and the respective phrase table sizes. For both of the test sets, the top three results were obtained in *baseline*, *sigfilter 30* and *eppex 1-in* experiments, but all the differences are rather small and could be also attributed to the randomness of MERT.[7] The *baseline* scenario ranked best on wmt11 set (BLEU score 18.22), while the *eppex 1-in* scenario topped on wmt10 set

[7]We did not invest the computing resources necessary to estimate the confidence bounds covering optimizer instability (Clark et al., 2011).

	Final phrase table size		BLEU score	
Experiment	phrase pairs	.gz file size	wmt10	wmt11
baseline	153.6 M	3.68 GB	17.36	**18.22**
sigfilter 30	137.0 M	3.36 GB	17.48	18.13
sigfilter a-e	92.4 M	2.39 GB	17.23	17.87
eppex 1-in	57.1 M	1.28 GB	**17.60**	18.10
sigfilter a+e	35.0 M	0.86 GB	17.31	17.99
eppex 1-out	14.4 M	0.33 GB	17.23	17.94

Table 2. Phrase table sizes and BLEU scores for all experiments

(BLEU score 17.60). However, the phrase table extracted in *eppex 1-in* occupied only 1.28 GB space on disk, being less than half of the size of the *baseline* (3.68 GB) and *sigfilter 30* (3.36 GB) phrase tables.

Harsher pruning in both *eppex* and *sigfilter* can reduce the phrase table size up to one tenth of the baseline with only negligible loss in BLEU.

4.2. Memory and time requirements

Table 3 presents in detail physical memory peaks for all experiments. In *eppex* scenarios, it is the epochal extraction itself that is the most demanding part of the pipeline, consuming 19.2 GB (*1-in*) and 16.7 GB (*1-out*) of memory. In all the other experiments the memory consumption is considerably lower and the *scorer* component is responsible for the peak, except for the cases when $\alpha \pm \epsilon$ significance filtering is applied.

Experiment	VM peak	in step
baseline	1.1 GB	scoring-e2f
sigfilter 30	1.1 GB	scoring-e2f
sigfilter a-e	5.4 GB	sigfilter
eppex 1-in	19.2 GB	phr-ext
sigfilter a+e	5.4 GB	sigfilter
eppex 1-out	16.7 GB	phr-ext

Table 3. Virtual memory peaks for all experiments

Table 4 compares CPU usage and wallclock times of all the substeps of the pipeline in *baseline* and *eppex* scenarios. While the initial extraction of phrase pairs takes much longer with *eppex* than with *extract*, subsequent steps finish much quicker in the *eppex* scenario: total time required is less than half in case of *1-in* pruning and less than 1/4

	baseline		eppex 1-in		eppex 1-out	
step	CPU	wallclock	CPU	wallclock	CPU	wallclock
phr-ext	1145	**1152**	4346	**4360**	4349	**4361**
gzip-f2e	590	662	183	226	86	114
gzip-e2f	593	641	185	276	87	132
sort-f2e	653	2257	304	822	196	564
sort-e2f	628	2844	315	810	199	567
score-f2e	10516	10795	4493	4531	371	372
score-e2f	9400	9622	2867	2902	340	340
sort-inv	358	1569	129	129	21	22
cons	754	1361	269	269	65	66
pt-gzip	791	881	258	259	65	65
TOTAL	25428	31784	13349	14584	5779	6603
hh:mm:ss	7:03:48	**8:49:44**	3:42:29	**4:03:04**	1:36:19	**1:50:03**

Table 4. CPU and wallclock times (in seconds) of the phrase table construction.

Sigf. settings	-l a+e		-l a-e		-n 30	
	CPU	wallclock	CPU	wallclock	CPU	wallclock
Sigfilter alone	18635	18248	19252	18449	2105	1141
TOTAL	44063	50032	44680	50233	27533	31784
hh:mm:ss	12:14:23	13:53:52	12:24:40	13:57:13	7:38:53	9:08:45

Table 5. CPU and wallclock times (in seconds) of significance filtering. The total includes the time of baseline extraction: 7:03:48 (CPU) and 8:49:44 (wallclock).

in case of *1-out* pruning compared to the baseline. Partial parallelization of the baseline extraction as suggested above decreases the gains but *eppex* still remains safely faster, esp. if *eppex* used the same optimization.

Significance filtering requires an additional amount of time (see Table 5) on top of the baseline extraction. The most striking difference is between *sigfilter* $\alpha+\epsilon$ and *eppex 1-out*. They are comparable in terms of BLEU score and phrase table size but *sigfilter* took almost 14 hours while *eppex 1-out* finished in less than 2 hours.

5. Related and Future Work

We point out that the idea of using approximate frequency counting algorithms in the field of NLP is not new. Goyal et al. (2009) used approximate n-gram frequency counts to build language models, which they then applied successfully in SMT achieving no significant loss in BLEU.

A recent feature of Moses is *incremental training* (probably related to the experiments by Levenberg et al. (2010)) aiming at the reduction of time required to incorporate recent data into already trained models. The entire source and target corpora are indexed in suffix arrays along with their alignments. Phrase extraction and scoring happens on the fly when phrases are needed in translation, completely eliminating the expensive batch retraining. Because the reduction of training time is the main advantage of *eppex*, we intend to perform a comprehensive comparison of this feature with batch retraining utilizing *eppex*.

The epochal extraction in *eppex* also lends itself to incremental extraction: only the counts in the current epoch have to be stored and reloaded when the model is to be extended by new data. The setting of the thresholds, however, would require a fairly large amount of training data to be processed in the first batch to estimate the values that will lead to sufficient saturation of phrase table. Our initial experiment suggests that for short phrase pair lengths it is beneficient to use no pruning at all.

Hardmeier (2010) presented *memscore*, an open-source tool to score phrases in memory that acts as a faster drop-in replacement for the sorting and the *scorer* in the pipeline. By combining *memscore* and *eppex* into a single phrase extraction tool a further speed up of phrase table construction process might be achieved.

Furthermore, we expect the benefits of *eppex* to be even more significant when confronted with larger training corpora, therefore we are in the process of its evaluation on the 10^9 French–English corpus available as part of WMT training data.

6. Conclusions

We presented *eppex*, a tool for extraction of phrase pairs from word-aligned parallel corpus capable of phrase pairs filtering on the fly based on a user-defined threshold. *Eppex* is a drop-in alternative of *extract* component in Moses training toolkit. Our tool is ready to use and it is available in Moses SVN trunk (in `scripts/training/eppex`).

We compared our tool against the baseline extraction and another common approach to phrase table filtration. By using *eppex* for phrase extraction we were able to obtain translation quality comparable to the baseline, while at the same time both the (wallclock) training time and phrase table size have been reduced by more than a half or up to one tenth with harsher pruning. Although memory requirements are significantly increased, they still lie within manageable limits.

7. Acknowledgment

This work has been supported by the grants EuroMatrixPlus (FP7-ICT-2007-3-231720 of the EU and 7E09003 of the Czech Republic), P406/10/P259, and MSM 0021620838.

Bibliography

Bojar, Ondřej and Zdeněk Žabokrtský. CzEng0.9: Large Parallel Treebank with Rich Annotation. *Prague Bulletin of Mathematical Linguistics*, 92:63–83, 2009. ISSN 0032-6585.

Clark, Jonathan H., Chris Dyer, Alon Lavie, and Noah A. Smith. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proc. of ACL/HLT*, pages 176–181, Portland, Oregon, USA, June 2011. URL `http://www.aclweb.org/anthology/P11-2031`.

Goyal, Amit, Hal Daumé, III, and Suresh Venkatasubramanian. Streaming for large scale NLP: language modeling. In *Proc. of HTL/NAACL*, pages 512–520, Boulder, Colorado, 2009. URL `http://portal.acm.org/citation.cfm?id=1620754.1620829`.

Hardmeier, Christian. Fast and Extensible Phrase Scoring for Statistical Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, 93:79–88, 2010.

Johnson, J Howard, Joel Martin, George Foster, and Roland Kuhn. Improving Translation Quality by Discarding Most of the Phrasetable. In *Proc. of EMNLP and Computational Natural Language Learning*, pages 967–975, 2007.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proc. of ACL (Demonstration Session)*, pages 177–180, 2007.

Levenberg, Abby, Chris Callison-Burch, and Miles Osborne. Stream-based translation models for statistical machine translation. In *Proc. of HTL/NAACL*, pages 394–402, Los Angeles, California, 2010. URL `http://portal.acm.org/citation.cfm?id=1857999.1858061`.

Manku, Gurmeet Singh and Rajeev Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

Mareček, David, Rudolf Rosa, Petra Galuščáková, and Ondřej Bojar. Two-step translation with grammatical post-processing. In *Proc. of WMT*, Edinburgh, UK, July 2011.

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

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pages 311–318, Philadelphia, Pennsylvania, 2002. URL `http://dx.doi.org/10.3115/1073083.1073135`.

Address for correspondence:
Ondřej Bojar
`bojar@ufal.mff.cuni.cz`
UK MFF ÚFAL
Malostranské náměstí 25
118 00 Praha 1, Czech Republic