PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 102 OCTOBER 2014 17–26

A Fast and Simple Online Synchronous Context Free Grammar Extractor

Paul Baltescu, Phil Blunsom

University of Oxford, Department of Computer Science

Abstract

Hierarchical phrase-based machine translation systems rely on the synchronous context free grammar formalism to learn and use translation rules containing gaps. The grammars learned by such systems become unmanageably large even for medium sized parallel corpora. The traditional approach of preprocessing the training data and loading all possible translation rules into memory does not scale well for hierarchical phrase-based systems. Online grammar extractors address this problem by constructing memory efficient data structures on top of the source side of the parallel data (often based on suffix arrays), which are used to efficiently match phrases in the corpus and to extract translation rules on the fly during decoding. This paper describes an open source implementation of an online synchronous context free grammar extractor. Our approach builds on the work of Lopez (2008a) and introduces a new technique for extending the lists of phrase matches for phrases containing gaps that reduces the extraction time by a factor of 4. Our extractor is available as part of the `cdec` toolkit[1] (Dyer et al., 2010).

1. Introduction

Grammar extraction is the part of a machine translation pipeline responsible for finding the set of applicable translation rules in a word-aligned parallel corpus. Every time a machine translation system receives a sentence as input, the extractor is queried for the set of translation rules that match subphrases of the given sentence. The overall translation time depends on the extractor's ability to efficiently identify these rules. This paper introduces a fast and simple grammar extractor for hierarchical phrase based translation systems.

[1] Our code is available here: `https://github.com/redpony/cdec/tree/master/extractor`.

 Corresponding author: paul.baltescu@cs.ox.ac.uk
Cite as: Paul Baltescu, Phil Blunsom. A Fast and Simple Online Synchronous Context Free Grammar Extractor. The Prague Bulletin of Mathematical Linguistics No. 102, 2014, pp. 17–26. doi: 10.2478/pralin-2014-0010.

The traditional approach to grammar extraction is achieved with the help of phrase tables, dictionary-like data structures that map all the source phrases in the training corpus to their target side candidates. Phrase tables are generated in a preprocessing step, by iterating over each sentence in the word-aligned parallel corpus and extracting all phrase pairs up to a fixed width, such that a translation rule contains a word only if it also contains all the words aligned to it (Och and Ney, 2004). Phrase tables have been designed with phrase-based systems in mind (Koehn et al., 2003; Och and Ney, 2004), where the number of extractable phrase pairs is linear in the phrase width parameter. Even so, loading all the translation rules into memory can be problematic for large corpora or in memory constrained environments (mobile devices, commodity machines, etc.).

Hierarchical phrase-based translation systems (Chiang, 2007) learn and use translation rules containing gaps. For such systems, the number of extractable translation rules is exponential in the phrase width parameter. As a result, the grammars learned by hierarchical phrase-based systems are too large to fit in memory for almost all relevant setups. A naive solution is to filter the phrase tables and remove all translation rules that are not applicable for a given test set, but this approach does not scale to unseen sentences which are to be expected by any machine translation application.

Several memory efficient alternatives to phrase tables have been proposed. Zens and Ney (2007) store phrase tables on disk organized in a prefix tree data structure for efficient read access. Callison-Burch et al. (2005) and Zhang and Vogel (2005) introduce a phrase extraction technique based on suffix arrays which extracts translation rules on the fly during decoding. Lopez (2007) shows how online extractors based on suffix arrays can be extended to handle phrases with gaps. These two approaches have comparable lookup times despite the fact that the former has a better asymptotic complexity (constant vs. logarithmic) because slower disk reads are involved. In most scenarios, the suffix array approach is preferred (e.g. Schwartz and Callison-Burch (2010)) because it yields several benefits. The phrase width limitation for translation rules is no longer required as it has no effect on the memory footprint of the precomputed data structures. Also, less time is spent when tuning translation models, as the precomputed data structures need not be constructed again when new scoring features are added.

The remainder of this paper describes our suffix array based grammar extractor for hierarchical phrase-based systems. Section 2 reviews how suffix arrays are used for contiguous phrase extraction (Lopez, 2008a). Section 3 introduces our new technique for extracting phrases with gaps. Section 4 briefly covers the intended usage for our tool and discusses other implementation specific details which might make our tool appealing to the research community. Section 5 concludes the paper with a set of experiments demonstrating the benefits of the novel technique introduced in this paper and other speed gains obtained as a result of careful implementation.

2. Grammar extraction for contiguous phrases

A suffix array (Manber and Myers, 1990) is a memory efficient data structure which can be used to efficiently locate all the occurrences of a pattern, given as part of a query, in some larger string (referred to as *text* in the string matching literature, e.g. Gusfield (1997)). A suffix array is simply the list of suffixes in the text string sorted in lexicographical order. A suffix is encoded by its starting position and the overall size of the suffix array is linear in the size of text string. A crucial property of suffix arrays is that all suffixes starting with a given prefix form a compact interval within the suffix array.

Suffix arrays are well suited to solve the central problem of contiguous phrase extraction: efficiently matching phrases against the source side of the parallel corpus. Once all the occurrences of a certain phrase are found, candidate translation rules are extracted from a subsample of phrase matches. The rule extraction algorithm (Och and Ney, 2004) is linear in the size of the phrase pattern and adds little overhead to the phrase matching step.

Before a suffix array can be applied to the phrase matching problem, the source side of the parallel corpus is preprocessed as follows: first, words are replaced with numerical ids and then all sentences are concatenated together into a single array. The suffix array is constructed from this array. In our implementation, we use a memory efficient suffix array construction algorithm proposed by Larsson and Sadakane (2007) having $O(N \log N)$ time complexity. The memory requirements of the suffix array are linear in the size of the training data.

The algorithm for finding the occurrences of a phrase in the parallel corpus uses binary search to locate the interval of suffixes starting with that phrase pattern in the suffix array. Let $w_1, w_2, \ldots, w_K$ be the phrase pattern. Since a suffix array is a sorted list of suffixes, we can binary search the interval of suffixes starting with w_1. This contiguous subset of suffix indices continues to be lexicographically sorted and binary search may be used again to find the subinterval of suffixes starting with w_1, w_2. However, all suffixes in this interval are known to start with w_1, so it is sufficient to base all comparisons on only the second word in the suffix. The algorithm is repeated until the whole pattern is matched successfully or until the suffix interval becomes empty, implying that the phrase does not exist in the training data. The complexity of the phrase matching algorithm is $O(K \log N)$. We note that $w_1, \ldots, w_{K-1}$ is a subphrase of the input sentence as well and the extractor applies the phrase matching algorithm for $w_1, \ldots, w_{K-1}$ as part of a separate query. Matching $w_1, \ldots, w_{K-1}$ executes the first $K-1$ steps of the phrase matching algorithm for $w_1, \ldots, w_K$. Therefore, the complexity of the matching algorithm can be reduced to $O(\log N)$ per phrase, by caching the suffix array interval found when searching for $w_1, \ldots, w_{K-1}$ and only executing the last step of the algorithm for $w_1, \ldots, w_K$.

Let M be the length of a sentence received as input by the decoder. If the decoder explores the complete set of contiguous subphrases of the input sentence, the suffix

array is queried $O(M^2)$ times. We make two trivial observations to further optimize the extractor by avoiding redundant queries. These optimizations do not lead to major speed-ups for contiguous phrase extraction, but are important for laying the foundations of the extraction algorithm for phrases containing gaps. First, we note that if a certain subphrase of the input sentence does not occur in the training corpus, any phrase spanning this subphrase will not occur in the corpus as well. Second, phrases may occur more than once in a test sentence, but all such repeating occurrences share the same matches in the training corpus. We add a caching layer on top of the suffix array to store the set of phrase matches for each queried phrase. Before applying the pattern matching algorithm for a phrase $w_1, \ldots, w_K$, we verify if the cache does not already contain the result for $w_1, \ldots, w_K$ and check if the search for $w_1, \ldots, w_{K-1}$ and $w_2, \ldots, w_K$ returned any results. The caching layer is implemented as a prefix tree with suffix links and constructed in a breadth first manner so that shorter phrases are processed before longer ones (Lopez, 2008a).

3. Grammar extraction for phrases with gaps

Synchronous context free grammars are the underlying formalism which enable hierarchical translation systems to use translation rules containing gaps. For a detailed introduction to synchronous context free grammars in machine translation see Lopez (2008b). In this section, we present an algorithm for extracting synchronous context free rules from a parallel corpus, which requires us to adapt the phrase extraction algorithm from Section 2 to work for discontiguous phrases.

Let us make some notations to ease the exposition of our phrase extraction algorithm. Let a, b and c be words in the source language, X a non-terminal used to denote the gaps in translation rules and α and β source phrases containing zero or more occurrences of X. Let M_α be the set of matches of the phrase α in the source side of the training corpus, where a phrase match is defined by a sequence of indices marking the positions where the contiguous subphrases of α are found in the training data. Our goal is to find M_α for every phrase α. Section 2 shows how to achieve this if X does not occur in α.

Let us consider the case when α contains at least one non-terminal. If $\alpha = X\beta$ or $\alpha = \beta X$, then $M_\alpha = M_\beta$, because the phrase matches are defined only in terms of the indices where the contiguous subpatterns match the training data. The words spanned by the leading or trailing non-terminal are not relevant because they do not appear in the translation rule. Since $|\beta| < |\alpha|$, M_β is already available in the cache as a consequence of the breadth first search approach we use to compute the sets M.

The remaining case is $\alpha = a\beta c$, where both $M_{a\beta}$ and $M_{\beta c}$ have been computed at a previous step. We take into consideration two cases depending on whether the next-to-last symbol of α is a terminal or not (i.e. $\alpha = a\beta bc$ or $\alpha = a\beta Xc$, respectively). In the former case, we calculate M_α by iterating over all the phrase matches in $M_{a\beta b}$ and selecting those matches that are followed by the word c. In the sec-

ond case, we take note of the experimental results of Lopez (2008a) who shows that translation rules that span more than 15 words have no effect on the overall quality of translation. In our implementation, we introduce a parameter `max_rule_span` for setting the maximum span of a translation rule. For each phrase match in $M_{\alpha\beta X}$, we check if any of the following `max_rule_span` words is c (subject to sentence boundaries and taking into account the current span of $\alpha\beta X$) and insert any new phrase matches in M_α accordingly. Note that M_α can also be computed by considering two cases based on the second symbol in α (i.e. $\alpha = ab\beta c$ or $\alpha = aX\beta c$) and by searching the word a at the beginning of the phrase matches in $M_{b\beta c}$ or $M_{X\beta c}$. In our implementation, we consider both options and apply the one that is likely to lead to a smaller number of comparisons. The complexity of the algorithm for computing $M_{\alpha=a\beta c}$ is $O(\min(|M_{a\beta}|, |M_{\beta c}|))$.

Lopez (2007) presents a similar grammar extraction algorithm for discontiguous phrases, but the complexity for computing M_α is $O(|M_{a\beta}| + |M_{\beta c}|)$. Lopez (2007) introduces a separate optimization based on double binary search (Baeza-Yates, 2004) of time complexity $O(\min(|M_{a\beta}|, |M_{\beta c}|) \log \max(|M_{a\beta}|, |M_{\beta c}|))$, designed to speed up the extraction algorithm when one of the lists is much shorter than the other. Our approach is asymptotically faster than both algorithms. In addition to this, we do not require the lists M_α to be sorted, allowing for a much simpler implementation. (Lopez (2007) needs van Emde Boas trees and an inverted index to efficiently sort these lists.)

The extraction algorithm can be optimized by precomputing an index for the most frequent discontiguous phrases (Lopez, 2007). To construct the index, we first need to identify the set of the most frequent K contiguous phrases in the training data, where K is an argument for our extraction tool. We use the LCP array (Manber and Myers, 1990), an auxiliary data structure constructed in linear time from a suffix array (Kasai et al., 2001), to find all the contiguous phrases in the training data that occur above a certain frequency threshold. We add these phrases to a max-heap together with their frequencies and extract the most frequent K contiguous patterns. We iterate over the source side of the training data and populate the index with all the discontiguous phrases of the form uXv and $uXvXw$, where u, v and w are amongst the most frequent K contiguous phrases in the training data.

4. Usage and implementation details

Our grammar extractor is designed as a standalone tool which takes as input a word-aligned parallel corpus and a test set and produces as output the set of translation rules applicable to each sentence in the test set. The extractor produces the output in the format expected by the `cdec` decoder, but the implementation is self-contained and easily extendable to other hierarchical phrase-based translation systems.

Our tool performs grammar extraction in two steps. The preprocessing step takes as input the parallel corpus and the file containing the word alignments and writes to disk binary representations of the data structures needed in the extraction step: the

symbol table, the source suffix array, the target data array, the word alignment, the precomputed index of frequent discontiguous phrases and a translation table storing estimates for the conditional word probabilities $p(s|t)$ and $p(t|s)$, for every source word s and target word t collocated in the same sentence pair in the training data. The translation probabilities are required for the scoring features in the extraction step. The output of the preprocessing step is written to disk in a directory specified by the user. A configuration file is also produced to reduce the number of parameters the user has to provide to the extraction step. The preprocessed data structures can be reused when extracting grammars for different test sets. The extraction step takes as input the precomputed data structures and a test corpus and produces a set of grammar files containing the applicable translation rules for each sentence in the test set. Note that our extraction tool expects the entire test corpus as input only to match the intended overall usage of the `cdec` pipeline and that the tool itself at no point takes advantage of the fact that the whole test corpus is known in advance.

Our extractor is written in `C++`. Compiling the code yields two binaries, `sacompile` and `extract`, corresponding to the two steps described above. `sacompile` takes the following parameters:

- `--help`: Prints a list of available options.
- `--source`: The path to the file containing the source side of the parallel corpus, one sentence per line.
- `--target`: The path to the file containing the target side of the parallel corpus, one sentence per line.
- `--bitext`: The path to the parallel corpus, one pair of sentences per line. The expected format is `source_sentence ||| target_sentence`. This parameter needs to be set only if `--source` and `--target` are not provided.
- `--alignment`: The path to the word alignment file. The expected format is the same as the one used by tools like `cdec` or `Moses` [2].
- `--output`: The directory where the binary representations are written.
- `--config`: The path where the config file will be created.
- `--max_rule_span`: The maximum number of words spanned by a rule.
- `--max_symbols`: The maximum number of symbols (words and non-terminals symbols) in the source side of a rule.
- `--min_gap_size`: The minimum number of words spanned by a non-terminal.
- `--frequent`: The number of frequent contiguous phrases to be extracted for the construction of the precomputed index.
- `--super_frequent`: The number of super frequent contiguous phrases to be used in the construction of the precomputed index (a subset of the contiguous phrases extracted with the `--frequent` parameter). Discontiguous phrases of the form $uXvXw$ are added to the index only if either both u and v or v and w are super-frequent.

[2] More details here: `http://www.cdec-decoder.org/guide/fast_align.html`

- `--min_frequency`: The minimum number of times a phrase must occur in the corpus to be considered a candidate for the set of most frequent phrases.
- `--max_phrase_len`: The maximum number of words spanned by a frequent contiguous phrase.

The `extract` binary takes the following parameters:
- `--help`: Prints a list of available options.
- `--config`: The path to the configuration file produced by the preprocessing step.
- `--grammars`: The directory where the files containing the translation rules for each sentence are written.
- `--threads`: The number of threads used for parallel extraction.
- `--max_rule_span`: The maximum number of words spanned by a rule.
- `--max_rule_symbols`: The maximum number of symbols (words and non-terminal symbols) in the source side of a rule.
- `--min_gap_size`: The minimum number of words spanned by a non-terminal.
- `--max_nonterminals`: The maximum number of non-terminals in a rule.
- `--max_samples`: A threshold on the number of phrase matches used to extract translation rules for each phrase.
- `--tight_phrases`: Use tight constraints for extracting rules (Chiang, 2007).
- `--leave_one_out`: If the training set is used as a test set, the extractor will ignore any phrase matches in the test sentence for which the rules are extracted.

The `extract` binary reads the test corpus from standard input and produces an summary file at standard output. For both binaries, the only required parameters are the files and directories required for input and output, while the remaining parameters are initialized with sensible default values.

Our implementation leverages the benefits of a multithreaded environment to speed up grammar extraction. The test corpus is distributed dynamically across the number of available threads (specified by the user with the `--threads` parameter). All the data structures computed in the preprocessing step are immutable during extraction and can be effectively shared across multiple threads at no additional time or memory cost. In contrast, the existing extractor (implementing Lopez (2008a)'s algorithm) available in `cdec` uses a multi-process approach to parallel extraction. This is ill-suited for memory constrained environments because the preprocessed data structures are copied across all the processes used for extraction. As a result, the amount of memory available will restrict the degree of parallelization that the extractor can achieve.

Our code is released together with a suite of unit tests based on the `Google Test` and `Google Mock` frameworks. The unit tests are provided to encourage developers to add their own features to our grammar extractor without the fear that their changes might have unexpected consequences.

Implementation	Time (minutes)	Memory (GB)
Original `cython` extractor	28.518	6.4
`C++` reimplementation	2.967	6.4
Current work (`C++`)	2.903	6.3

Table 1. Results for the preprocessing step.

Implementation	Time (minutes)	Memory (GB)
Original `cython` extractor	309.725	4.4
`C++` reimplementation	381.591	6.4
Current work (`C++`)	75.496	5.7

Table 2. Results for the phrase extraction step.

5. Experiments

In this section, we present a set of experiments which illustrate the benefits of our new extractor. We compare our implementation with the one available in `cdec` which implements the algorithm proposed by Lopez (2008a). The existing extractor is written in `cython`. In order to make the comparison fair and to prove that the speed ups we obtain are indeed a result of our new algorithm, we also report results for an implementation of Lopez (2008a)'s algorithm in `C++`.

For our experiments, we used the French-English data from the `europarl-v7` corpus, a set of 2,002,756 pairs of sentences containing a total of 104,722,300 tokens. The training corpus was tokenized, lowercased and pairs of sentences with unusual length ratios were filtered out using the corpus preparation scripts available in `cdec`[3]. The corpus was aligned using `fast_align` (Dyer et al., 2013) and the alignments were symmetrized using the `grow-diag-final-and` heuristic. We extracted translation rules for the `newstest2012` test corpus[4]. The test corpus consists of 3,003 sentences and was tokenized and lowercased using the same scripts as the training corpus.

Table 1 shows results for the preprocessing step of the three implementations. We note a 10-fold time reduction when reimplementing Lopez (2008a)'s algorithm in `C++`. We believe this is a consequence of inefficient programming when the precomputed index is constructed in the `cython` code and not a result of using different programming languages. Our new implementation does not significantly outperform an efficient implementation of the preprocessing step of Lopez (2008a)'s extractor because it computes the same set of data structures.

[3]We followed the indications provided here: `http://www.cdec-decoder.org/guide/tutorial.html`.

[4]The test corpus is available here: `http://www.statmt.org/wmt14/translation-task.html`.

Implementation	Time (minutes)	Memory (GB)
Original `cython` extractor	37.950	35.2
`C++` reimplementation	51.700	10.1
Current work (`C++`)	9.627	6.1

Table 3. Results for parallel extraction using 8 processes/threads.

The second set of results (Table 2) show the running times and memory requirements of the extraction step. Our `C++` reimplementation of Lopez (2008a)'s algorithm is slightly less efficient than the original `cython` extractor, supporting the idea that the two programming languages have roughly similar performance. We note that our novel extraction algorithm is over 4 times faster than the original approach of Lopez (2008a). The increased memory usage is not a real concern because it does not exceed the amount of memory used in the preprocessing step.

Table 3 demonstrates the benefits of parallel phrase extraction. We repeated the experiments from Table 2 using 8 processes in `cython` and 8 threads in `C++`. As expected, the running times decrease roughly 8 times. The benefits of shared-memory parallelism are evident, our new implementation is saving 29.1 GB of memory. Our implementation continues to use less memory than the preprocessing step even when running in multithreaded mode.

In conclusion, this paper presents an open source implementation of a SCFG extractor integrated with `cdec` that is 4 times faster than the existing extractor (Lopez, 2008a) and that is better designed for parallel environments. Compared to traditional phrase tables, our approach is considerably more memory efficient without involving any pruning based on the test corpus, therefore scaling to unseen sentences.

Bibliography

Baeza-Yates, Ricardo A. A fast set intersection algorithm for sorted sequences. In *Combinatorial Pattern Matching*, pages 400–408. Springer Berlin Heidelberg, 2004.

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

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

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.

Dyer, Chris, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL '13)*, pages 644–648, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

Gusfield, Dan. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, USA, 1997.

Kasai, Toru, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching*, pages 181–192. Springer Berlin Heidelberg, 2001.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL '03)*, pages 48–54. Association for Computational Linguistics, 2003.

Larsson, N. Jesper and Kunihiko Sadakane. Faster suffix sorting. *Theoretical Computer Science*, 387(3):258–272, 2007.

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 '07)*, pages 976–985, Prague, Czech Republic, 2007. Association for Computational Linguistics.

Lopez, Adam. *Machine translation by pattern matching*. ProQuest, 2008a.

Lopez, Adam. Statistical machine translation. *ACM Computing Surveys*, 40(3):1–49, 2008b.

Manber, Udi and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 319–327. Society for Industrial and Applied Mathematics, 1990.

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

Schwartz, Lane and Chris Callison-Burch. Hierarchical phrase-based grammar extraction in joshua. *The Prague Bulletin of Mathematical Linguistics*, 93(1), 2010.

Zens, Richard and Hermann Ney. Efficient phrase-table representation for machine translation with applications to online MT and speech translation. In *Proceedings of the 2007 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL '07)*, pages 492–499, Rochester, New York, 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 30–31, 2005.

Address for correspondence:
Paul Baltescu
paul.baltescu@cs.ox.ac.uk
Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom