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

Unsupervised Generation of Parallel Treebanks through Sub-Tree Alignment

Ventsislav Zhechev

Abstract

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

1. Motivation

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

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

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

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

2. Algorithms

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

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

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

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

2.1. Tree-to-Tree Alignment

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

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

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

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

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

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

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

2.1.1. Translational Equivalence

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

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

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

(3) score1
$$\alpha(x|y) = \prod_j^{|y|} \sum_i^{|x|} P(x_i|y_j)$$

(4) score2
$$\alpha(x|y) = \prod_i^{|x|} \frac{\sum_j^{|y|} P(x_i|y_j)}{|y|}$$

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

2.1.2. Greedy-Search Algorithm

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

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

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

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

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

2.1.3. Full-Search Algorithm

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

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

2.2. Other Alignment Modules

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

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

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

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

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

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

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

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

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

2.3. Re-scoring

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

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

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

3. Usage

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

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

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

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

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

3.1. Tree-to-Tree Aligner

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

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

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

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

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

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

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

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

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

```
   align <source_to_target_lex_probs> <target_to_source_lex_probs> [<source_to_tar-
get_phrase_probs>] <input_corpus>
   align <config_file>
```

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

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

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

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

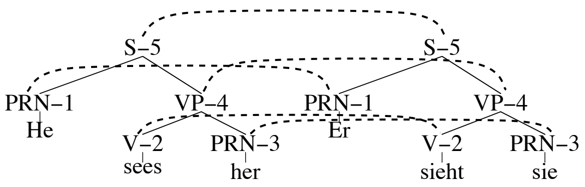
<input_corpus> The path to the file containing the aligned parsed sentences or –. Supplying – for this parameter will direct the aligner to read data from the standard input, rather than from a file. The format is <source>\n<target>\n\n\n. The parsed sentences should be in bracketed format, using (and)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

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

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

If you compile the lattice tool, it will use the exact same options as the tree-to-tree aligner and will produce output in the same format. The most significant difference is that it will use the full-search algorithm for the induction of the sub-tree alignments, rather than the greedy-search based algorithm. This tool is still experimental, though, and due to the combinatorial nature of the full-search algorithm may not find a solution for all sentence pairs within an acceptable timeframe. Because of this the use of this tool is strongly discouraged. The `lattice` tool does not support the *span1* extension yet and the *skip** modules are irrelevant to it.

```
(S (PRN He)(VP (V sees)(PRN her)))
(S (PRN Er)(VP (V sieht)(PRN sie)))
```



```
(S-5 (PRN-1 He)(VP-4 (V-2 sees)(PRN-3 her)))
(S-5 (PRN-1 Er)(VP-4 (V-2 sieht)(PRN-3 sie)))
1 1 2 2 3 3 4 4 5 5
```

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

3.2. String-to-String Aligner

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

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

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

Here is the description of the options:

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

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

`<output_type>` This argument is used to select the type of output of the aligner and cannot be omitted. There are three possible options: `standard`, `parse` and `XML`. The `standard` output has the format presented in Figure 2 for each sentence pair. If the `mother_node_ID` of a node is `0`, then this node has no ancestors (it is a root node). There may be more than one such node for each sentence. This format preserves enough nodes to represent all possible binary trees for the sentences in the pair that are consistent with the induced links. The parse and XML output formats present minimal trees, consisting only of the pre-terminal nodes and the linked nodes for the sentences in each pair. In case there is more than one root node for a particular tree, an extra node with label *X* and ID 100000 is inserted as the mother of all root nodes. Both formats give a standard bracketed representation of unambiguous parse trees.

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

#BOP
…
#EOP
```

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

<input_corpus> There are two possible formats for the sentences, while the overall file format remains as for the tree-to-tree aligner. The first format is simply <word$_1$> <word$_2$> … <word$_n$>. The second format is ((<word$_1$>)) ((<word$_2$>)) … ((<word$_n$>)) and can be used to specify the boundaries of multiword units. This second format can also be used for supplying POS tags for the words of the sentences. In that case the format is ((<word$_1$> <POS$_1$>)) ((<word$_2$> <POS$_2$>)) … ((<word$_n$> <POS$_n$>)). A specific requirement for the use of the string-to-string aligner is the existence of one of two open source modules on your system: If you are using the first input format, you need the Boost Tokenizer library; If you are using the second input format, you need the Boost Regex library.

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

4. Conclusion

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

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

Acknowledgments

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

Bibliography

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

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

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

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

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

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

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

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