The IMS Open Corpus Workbench (CWB)
Corpus Encoding Tutorial

— CWB Version 3.4.24 —

Stefan Evert & The CWB Development Team
http://cwb.sourceforge.net/

May 2020

Contents

1 Prerequisites

In order to follow this tutorial, you need to install the **IMS Open Corpus Workbench (CWB)**, version **3.0** or newer, which can be downloaded from

http://cwb.sourceforge.net/download.php

It is easiest to install a pre-compiled **binary package**, following instructions on the Web page and in the enclosed `README` file. You should also install the **CWB/Perl interface**, which includes the useful `cwb-make` and `cwb-regedit` programs. A **data package** with all input files needed for the examples in this tutorial below is available from

http://cwb.sourceforge.net/documentation.php

2 First steps: Encoding and indexing

The standard CWB **input format** is one-word-per-line text,[1] with the surface form in the first column and token-level annotations specified as additional `TAB`-separated columns. XML **tags** for sentence boundaries and other structural annotation must appear on separate lines. This file format is also called **verticalized text** and has the customary file extension `.vrt`. An example of the verticalized text format for a short sentence with part-of-speech and lemma annotations is shown in Figure 1. This file, as well as all other input files required by the following examples are made available in the accompanying **data package**.

```
<s>
It        PP      it
was       VBD     be
an        DT      an
elephant  NN      elephant
.         SENT    .
</s>
```

Figure 1: Verticalized text file *example.vrt*

In order to encode the file as a corpus, follow these steps:

- Create a **data directory** where files in the binary CWB format will be stored. Here, we assume that this directory is called `/corpora/data/example`. If this directory already exists contains corpus data (from a previous version), you should delete all files in the directory. **NB:** You need a separate data directory for each corpus you want to encode.
- Choose a **registry directory**, where all encoded corpora have to be registered to make them accessible to the CWB tools. It is recommended that you use the **default registry** directory `/usr/local/share/cwb/registry`.[2] Otherwise, you will have to specify the path to your registry

[1] Or, more precisely, one token per line; i.e., CWB expects punctuation marks, parentheses, quotes, etc. on separate lines. The precise **tokenization rules** depend on your theoretical assumptions and the requirements of annotation software such as part-of-speech taggers. Note that the CWB does *not* include any NLP components and has to be provided with a tokenized and annotated corpus.

[2] In previous versions of CWB, the default registry directory used to be `/corpora/c1/registry` (for historical reasons). All binary packages of CWB 3.0 and newer use the new default setting. If you already have a working environment with the old registry path, you may want to compile the CWB source code yourself, selecting the `traditional` site configuration.

directory with a `-r` flag whenever you invoke one of the CWB tools (or set an appropriate environment variable, see below). In this tutorial, we assume that you use the standard registry directory.

- The next step is to **encode** the corpus, i.e. convert the verticalized text to CWB binary format with the `cwb-encode` tool. Note that the command below has to be entered on a single line.

```
$ cwb-encode -d /corpora/data/example -f example.vrt
             -R /usr/local/share/cwb/registry/example
             -P pos -P lemma -S s
```

(The `$` character indicates a command line to be entered in a shell. It is inspired by the customary input prompt used by the Bourne shells `sh` and `bash`.)

The first column of the input file is automatically encoded as the default **positional attribute** (**p-attribute**) named `word`. `-P` flags are used to declare additional p-attributes, i.e. token-level annotations. `-S` flags declare **structural attributes** (**s-attributes**), which encode *non-recursive* XML tags and whose names must correspond to the XML element names. By convention, all attribute names must be *lowercase* (more precisely, they may only contain the characters `a-z`, `0-9`, `-`, and `_`, and may not start with a digit). Therefore, the names of XML elements to be included in the CWB corpus must not contain any non-ASCII or uppercase letters.

The `-R` option automatically creates a **registry file**, whose filename has to be written in *lowercase*. Note that it is necessary to specify the full path to the registry file, even if the default registry directory is used. The **CWB name** of the corpus (also called the corpus ID) is identical to the name of the registry file, but is written in *uppercase* (here it will be `EXAMPLE`). The CWB name is used to activate a corpus in the query processor CQP, for instance.

Input files with the extension `.gz` are assumed to be in gzip format and are automatically decompressed (provided that the `gzip` program is installed on your computer). Multiple input files can be specified by using the `-f` option repeatedly, and will be read in the order in which they appear on the command line. Note that shell wildcards (e.g. `-f *.txt`) do not work (since each file name must be preceded by `-f`). However, it is possible to read *all* files named `*.vrt` or `*.vrt.gz` in a given directory using the `-F` option (possibly repeated for multiple directories). Input files from the same directory will be read in alphabetical order.

As of CWB v3.4.11, files in `.bz2` format are also supported (provided that the `bzip2` program is installed), and compressed files are accepted for input and output by all CWB command-line tools.[3] Moreover, it is possible to read from or write to shell pipes in these versions, by specifying a quoted filename that starts with a pipe character (`|`).

All options (`-d`, `-f`, `-R`, etc.) *must precede* the attribute declarations (`-P`, `-S`, etc.) on the command line. It is mandatory to specify a data directory with the `-d` option.[4] This directory should always be given as an *absolute* path, so the corpus can be used from any location in the file system.

- Before a corpus can be used with CQP and other CWB programs, various **index files** have to be built. It is also strongly recommended to **compress** data files, especially for larger corpora.
- The easiest and recommended method for indexing and compression is to use the **cwb-make** script that comes with the **CWB/Perl** interface modules. If you are unable to install the modules and use this script, refer to Section 3 for a manual procedure.

[3] Previous CWB versions had partial support for gzip-compressed input and output files, indicated in the respective man pages.

[4] Previous versions of the CWB would default to the current working directory. As a result, simply typing `cwb-encode` on the command line would litter this directory with a number of empty data files and then hang, waiting for corpus data on the standard input.

```
$ cwb-make -V EXAMPLE
```

- If you do not use the standard registry directory `/usr/local/share/cwb/registry`, you will have to specify the path to your registry directory with the `-r` option. Alternatively, you can set the environment variable `CORPUS_REGISTRY`, which will automatically be recognized by all CWB programs. In a Bourne shell (`sh` or `bash`), this is achieved with the command

  ```
  $ export CORPUS_REGISTRY=/home/stefan/registry
  ```

  In a C shell (`csh` or `tcsh`), the corresponding command is

  ```
  $ setenv CORPUS_REGISTRY /home/stefan/registry
  ```

  It is probably a good idea to add this setting to your login profile (`~/.profile` or `~/.login`). If you do not want to set the environment variable, you need to invoke `cwb-make` with

  ```
  $ cwb-make -r /home/stefan/registry -V EXAMPLE
  ```

  The following examples assume that you either use the default registry directory or have set the `CORPUS_REGISTRY` variable appropriately.

- You can also specify multiple registry directories separated by colon characters (`:`), both in the `CORPUS_REGISTRY` environment variable and the `-r` options of command-line tools. This is convenient e.g. if some corpora are stored on external hard drives that are not always mounted. Such *optional* registry directories may be prefixed by a question mark (`?`) in order to indicate that they may not be accessible (otherwise CQP and some other tools will print warnings to alert you to possible typos in the registry path). For instance, one of the lead CWB developers has the following registry path in his `~/.bashrc` configuration:

  ```
  $ export CORPUS_REGISTRY=/Corpora/registry:?/Volumes/X/CWB/registry
  ```

  Note that the built-in default registry directory `/usr/local/share/cwb/registry` is not automatically appended to this path. If you want to specify additional registry directories but keep the default one, you will have to include it in the value of `CORPUS_REGISTRY`.

The `-V` switch enables additional validation passes when an index is created and when data files are compressed. It should be omitted when encoding very large corpora (above 50 million tokens), in order to speed up processing. In this case, it is also advisable to limit memory usage with the `-M` option. The amount specified should be somewhat less than the amount of physical RAM available (depending on the number of users etc.; too little is better than too much). For instance, on a Linux machine with 128 MB of RAM, `-M 64` is a safe choice. Note that the `cwb-make` utility applies a default limit of `-M 75` if not explicit `-M` option is given.

- Display some information about the encoded corpus (add `-s` option for details and to ascertain that all necessary data files have been created).

  ```
  $ cwb-describe-corpus EXAMPLE
  ```

3 Indexing and compression without CWB/Perl

If you do not have the CWB/Perl inteface installed, by far the best thing you can do is to install the CWB/Perl modules and the included scripts, and then go back to Section 2. If it is absolutely impossible to install CWB/Perl or you really want to learn the nitty-gritty of corpus encoding, continue here.

- In the manual procedure, indexing and compression are performed in separate steps by different tools. First, you have to run `cwb-makeall` in order to build the necessary index files.

  ```
  cwb-makeall -V EXAMPLE
  ```

  Note that `cwb-makeall` accepts the same `-V`, `-M` and `-r` options as `cwb-make`.

When the index files have been created, the corpus can already be used with CQP and other CWB tools. However, it is recommended that you compress the binary data files to save disk space and improve performance, especially for large corpora (above 10 million tokens). Compression is only supported for p-attributes at the current time.

- For positional attributes, both the token stream data and the index can be compressed. There are separate tools for compressing the two types of data files.
- The token stream can be compressed with the `cwb-huffcode` tool. Use the `-P` option to process a single attribute, or compress all p-attributes with `-A`.

  ```
  $ cwb-huffcode -A EXAMPLE
  ```

- Index files can be compressed with the `cwb-compress-rdx` tool, which accepts the same options.

  ```
  $ cwb-compress-rdx -A EXAMPLE
  ```

When compression was successful, both tools will display the full pathnames of uncompressed data files that are now redundant and can be deleted (namely, *attrib*`.corpus` after running `cwb-huffcode`, as well as *attrib*`.corpus.rev` and *attrib*`.corpus.rdx` after running `cwb-compress-rdx`).

If you run `cwb-makeall` again, it will show now that the p-attributes are compressed. Note that the compressed data files are validated by default, so it is safe to remove the redundant files. Validation can be turned off with the `-T` option, but is less performance-critical than with `cwb-makeall`.

- **NB:** If you re-encode a corpus, it is important to *erase all files* in the data directory first. The `cwb-makeall` program will not recognize that existing index files or compressed data files are out of date, and will therefore fail to rebuild them automatically. This is one of the reasons why the CWB/Perl `cwb-make` tool should be preferred.

4 CWB corpora and XML

Nowadays, machine-readable text and linguistic annotations are often provided in **XML format**. Version 3.0 of the IMS Open Corpus Workbench offers improved XML support, which is activated by the following encoding options: `-x` for XML compatibility mode (recognises default entities and skips comments as well as an XML declaration), `-s` to skip blank lines in the input, and `-B` to strip whitespace from tokens. The verticalized text format with `TAB`-separated p-attributes is still required by `cwb-encode`, but this format can easily be generated from an arbitrary XML file with the help of an XSLT stylesheet. Figure 2 shows a typical example of an XML input file for the CWB (note that this is still a well-formed XML file).

XML elements (i.e. matching pairs of start and end tags) can be encoded as s-attributes, which have to be declared with `-S` flags (for the file *vss.vrt*, the flags `-S story -S p -S s` would be used). If XML regions of the same type are **nested**, encoding will only work correctly if you add `:0` to the s-attribute declaration, which enables a rudimentary XML parser built into `cwb-encode`. **Attribute-value pairs** in XML start tags, such as

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!-- A Thrilling Experience -->
<story num="4" title="A Thrilling Experience">
<p>
<s>
Tick    NN      tick
.       SENT    .
</s>
<s>
A       DT      a
clock   NN      clock
.       SENT    .
</s>
<s>
Tick    VB      tick
,       ,       ,
tick    VB      tick
.       SENT    .
</s>
</p>
...
</story>
```

Figure 2: Verticalized XML file *vss.vrt*

```
<story num="4" title="A Thrilling Experience">
```

can be stored as a single unparsed text string (`num="4" title="A Thrilling Experience"`) by using the flag `-V` instead of `-S`. This form of encoding is not convenient for CQP queries, though. It is more desirable to declare XML tag attributes explicitly, which will automatically split the XML elements into multiple s-attributes. Note that the options `-xsB` should (almost) always be used and will automatically ignore the XML declaration and the comment line in Figure 2.

- Encode the verticalized XML file *vss.vrt* as a CWB corpus, with indexing and compression. **NB:** The last attribute declaration flag (`-0 collection`) is a *digit zero* (for a "null attribute", see below).

```
$ cwb-encode -d /corpora/data/vss -f vss.vrt
             -R /usr/local/share/cwb/registry/vss
             -xsB -P pos -P lemma
             -S s:0 -S p:0 -S story:0+num+title -0 collection

$ cwb-make -V VSS
```

  If you do not have the `cwb-make` script available, follow the steps in Section 3.

These commands will encode the corpus `VSS` and create a registry file, including the s-attributes `s`, `p`, `story`, `story_num`, and `story_title`. The `<story>` start tags are parsed and the attribute values are stored as annotations of the attributes `story_num` (value: `4`) and `story_title` (value: `A Thrilling Experience`). Regions of the `story` attribute itself will not be annotated. Use `-V` instead of `-S` to

store all attribute-value pairs as a single string, which can be useful for displaying and re-exporting the XML tags.

XML elements with different names (such as `<s>` and `<p>`) are encoded independently, so they can nest and overlap in arbitrary ways. The `cwb-encode` program does not perform any validation or well-formedness tests. When elements are nested recursively (e.g. a `<table>` within a `<table>`), the embedded elements will be ignored, though. After encoding, `cwb-encode` prints a summary listing the number of dropped XML elements. If you want to preserve nested elements, you can specify a maximal level of embedding instead of `:0` in the examples above. For instance, `-S table:2` allows two levels of embedding for `<table>` elements. Nested elements are automatically renamed to `<table1>` and `<table2>`, respectively, and stored in separate s-attributes.

Sometimes, the input data may contain XML tags that should not be encoded in the corpus. For instance, the stories in *vss.vrt* have to be wrapped in a single root element `<collection>` in order to obtain a well-formed XML file. Instead of removing such tags during data preparation, they can directly be filtered out by the `cwb-encode` tool. For this purpose, they have to be declared with the flag `-0` (digit zero, for "null attribute") instead of `-S` or `-V`. All start and end tags of these elements will be ignored completely. There is no need to add `:0` or XML attribute declarations. Note that all XML tags that have not been declared with a `-S`, `-V` or `-0` flag will be encoded as literal tokens (without annotations), accompanied by a warning message.

Starting with CWB 3.4.21, unknown XML tags can automatically be declared as null attributes with the `-9` ("auto-null") option. This is recommended to capture the correct token stream for an input file with very many and/or undocumented XML elements:

```
$ cwb-encode -d /corpora/data/vss -f vss.vrt
             -R /usr/local/share/cwb/registry/vss
             -xsB -c ascii -9 -P pos -P lemma
```

You may have noticed in Figure 2 that the XML file is declared to be in **ISO-8859-1** (or **Latin-1**) encoding rather than the standard UTF-8 format. **CWB version 3.0** has been developed exclusively for ISO-8859-1 data (which was still widely used for German corpus data until the mid-2010's, e.g. by TreeTagger and associated tools). While it is possible to store and query data in other ISO-8859-x encodings – or Unicode data in UTF-8 format – some features will not work properly unless the ISO-8859-1 encoding is used.[5] If you need to handle non-Latin-1 data with CWB 3.0, make sure that you are aware of the precise limitations involved.

CWB version 3.4 and later provide full support for all ISO-8859-x encodings as well as UTF-8. For these CWB versions, the recommended corpus encoding is UTF-8. Note that `cwb-encode` still defaults to Latin-1 (for backward compatibility) and a different encoding has to be specified with the `-c` option (e.g. `-c utf8` for Unicode data in UTF-8 format).

5 Adding attributes to an encoded corpus

In order to **add positional attributes** to a corpus that has already been encoded, create input data in the standard verticalized format, but listing only the new attributes. Figure 3 shows an example of such an input file, containing WordNet synonyms for the tokens from Figure 1 (without attempting any form of word sense disambiguation). A corresponding list of synonyms for the complete `VSS` corpus can be found in the file *syns.vrt*.

[5] In particular, case-insensitive (`%c`) and accent-insensitive (`%d`) matching as well as LaTeX notation for accented characters are only supported for ISO-8859-1 data. Regular expressions do not work properly for UTF-8 data and should only be used to express simple prefix and suffix constraint such as `.*able` in this case.

```
|
|be|cost|live|work|equal|exist|occur|...|
|
|elephant|
|
```

Figure 3: WordNet synonyms for the text shown in Figure 1 (excerpt from file *syns.vrt*)

The special notation seen in Figure 3 indicates that the synonyms for any given word constitute an unordered **set** (or **feature set** in CWB terminology). Vertical bars (`|`) separate individual set elements and enclose the entire set; a single bar `|` denotes the empty set. Feature sets are stored as plain strings in a CWB-encoded corpus, but the special notation enables the query processor CQP to test whether a particular string is contained in the set, match all set elements against a regular expression, and compute the intersection of two sets.

- The file *syns.vrt* is encoded as usual, but the default `word` attribute has to be *suppressed* with the option `-p -`. It is highly recommended to check that the number of tokens in the new file (`wc -l syns.vrt`) is equal to the corpus size (as reported by `cwb-lexdecode -S EXAMPLE`), so that the new attribute is properly aligned to the rest of the corpus.

  ```
  $ cwb-encode -d /corpora/data/vss -f syns.vrt -p - -P syn/
  ```

  Notice the slash (`/`) appended to the attribute name `syn`. This notation indicates that the new attribute should be treated as a feature set; `cwb-encode` will automatically validate and normalise the supplied values, issuing warnings if they are not well-formed feature sets. (A feature-set attribute that is not declared as such at index-time *can* still be treated as a feature set in CQP, but in this case responsibility is with the user to ensure that the values are well-formed feature sets.)

- The registry file for the corpus `VSS` (usually */usr/local/share/cwb/registry/vss*) has to be edited in order to declare the new attribute. Add the line

  ```
  ATTRIBUTE syn
  ```

  at the bottom of the file. If the CWB/Perl interface has been installed, the registry file can also be edited from the command line with the `cwb-regedit` registry editor script:

  ```
  $ cwb-regedit VSS :add :p syn
  ```

  This script can also be used to list and delete attributes, and to print basic information about a corpus (similar to `cwb-describe-corpus`, but easier for further processing). Type `cwb-regedit -h` for further information.

- Now you can build index files and compress the new attribute:

  ```
  $ cwb-make -V VSS
  ```

In order to **add structural attributes** with computed start and end points (**corpus positions**), you can use the `cwb-s-encode` tool. The corresponding start and end positions of existing s-attributes can be obtained with `cwb-s-decode`. The following example adds information about sentence length to the `VSS` corpus.

- The existing `s` attribute is decoded into a temporary file, then `awk` is used to compute sentence lengths, and the resulting annotated regions are encoded with `cwb-s-encode`.

```
$ cwb-s-decode VSS -S s > s.list
$ awk 'BEGIN { FS=OFS="\t" }  { print $1, $2, $2-$1+1 }' s.list > s_len.list
$ cwb-s-encode -d /corpora/data/vss -f s_len.list -V s_len
```

Note that it is currently *not necessary* to run `cwb-make` after adding an s-attribute.

- However, the new attribute still has to be declared in the registry file, either by manually adding

  ```
  STRUCTURE s_len
  ```

  or from the command line using the registry editor script:

  ```
  $ cwb-regedit VSS :add :s s_len
  ```

Tables of corpus positions as input for `cwb-s-encode` can also be created from CQP query results using the `dump` or `tabulate` command in a CQP session.

6 Adding XML annotations

In order to **add XML annotations** (e.g. `<np>` and `<pp>` tags inserted by a chunk parser) to an existing corpus, the usual strategy is to decode the token stream (and other attributes if necessary) to a temporary file. A chunk parser will often expect `<s>` and `</s>` tags marking sentence boundaries.

- Decode token stream (word forms) with start and end tags for `<s>` regions.

  ```
  $ cwb-decode -C VSS -P word -S s > word_s.vrt
  ```

- We then run the chunk parser on the temporary file, which adds its `<np>` and `<pp>` tags to the token stream, creating the file shown in Figure 4. This file is also provided as part of the **data package** for this tutorial.

```
<s>
<np head="experience>
My
experience
<pp head="of">
of
<np head="life">
life
</np>
</pp>
</np>
did
not
...
</s>
```

Figure 4: Decoded text with chunk annotations (file *chunks.vrt*)

- It is important that the token stream is left intact when adding XML annotations. In particular, tokens (as well as XML tags) must remain on separate lines and may not be split or combined. As a preliminary check, make sure that the number of tokens in *chunks.vrt* is equal to the corpus size.

```
$ grep -v '^<' chunks.vrt | wc -l
```

Now we can use `cwb-encode` to encode the XML annotations as structural attributes. The start and end points of regions are automatically computed from the token stream. Since we do not want to overwrite the `word` attribute, we specify `-p -`. With no p-attributes declared, all lines in the input file except for the XML tags will be ignored. Recall that `-0 s` (digit zero) instructs `cwb-encode` to ignore `<s>` and `</s>` tags (without `-S s` they would otherwise be interpreted as literal tokens and mess up the token stream).

- Encode `<np>` and `<pp>` regions in *chunks.vrt* as new s-attributes.

```
$ cwb-encode -d /corpora/data/vss -f chunks.vrt
             -p - -0 s -S np:0+head -S pp:0+head
```

In this example, `cwb-encode` will issue warnings about nested regions being dropped. As can be seen from Figure 4, `<np>` (as well as `<pp>`) regions may be embedded recursively. In order to preserve such nested regions, change the `:0` modifier to `:2`, allowing up to two levels of embedding (separately for each element type, i.e. `<np>` regions embedded in larger `<np>` regions, etc.). In general, `:`n allows up to n levels of embedding. The embedded regions will automatically be renamed to `np1`, `np2`, `pp1`, and `pp2`, respectively.

- Encode *chunks.vrt*, allowing up to two levels of embedding for `<np>` and `<pp>` regions.

```
$ cwb-encode -d /corpora/data/vss -f chunks.vrt
             -p - -0 s -S np:2+head -S pp:2+head
```

- The full list of s-attributes created by this command is `np`, `np1`, `np2`, `np_head`, `np_head1`, `np_head2`, `pp`, `pp1`, `pp2`, `pp_head`, `pp_head1`, and `pp_head2`. They have to be declared in the registry file of the corpus `VSS`, either by adding the appropriate entries manually, or with the registry editor script:

```
$ cwb-regedit VSS :add :s np np1 np2 np_head np_head1 np_head2
$ cwb-regedit VSS :add :s pp pp1 pp2 pp_head pp_head1 pp_head2
```

- Attribute-value pairs in XML start tags may also contain feature sets. For instance, the German chunk parser YAC[6] uses this notation to represent partially disambiguated morphological features of NPs and PPs (see the CQP Query Language Tutorial for more information and examples). XML tags of the form

  ```
  <np agr="|Nom:F:Sg|Acc:F:Sg|" head="Wiese">
  ```

  might be encoded with the declaration `-S np:2+agr/+head`, where the slash `/` indicates that `agr` values are feature sets. Since `head` is not followed by a slash, the corresponding values are not treated as feature sets.

7 Accessing frequency information

The `cwb-lexdecode` tool procides access to the **lexicon** of positional attributes, i.e. lists of all word forms or annotation strings (types) with their corpus frequencies. The `-S` option prints the size of corpus (*tokens*) and lexicon (*types*) only, `-P` selects the desired p-attribute, `-f` shows corpus frequencies, and `-s` lists the lexicon entries alphabetically (according to the internal sort order). In order to sort the lexicon by frequency, an external program (e.g. `sort`) has to be used.

[6] http://www.ims.uni-stuttgart.de/~kermes/YAC/YAC.shtml

```
$ cwb-lexdecode -S    -P lemma VSS
$ cwb-lexdecode -f -s -P lemma VSS | tail -20
$ cwb-lexdecode -f    -P lemma VSS | sort -nr -k 1 | head -20
```

It is also possible to annotate strings from a file (called *tags.txt* here) with corpus frequencies. The file must be in one-word-per-line format. `-0` (digit zero) prints a frequency of 0 for unknown strings rather than issuing a warning message; it can be combined with `-f` to the mnemonic form `-f0`.

```
$ cwb-lexdecode -f0 -P pos -F tags.txt VSS
```

With the `-p` option, word forms or annotations matching a regular expression can be extracted. Case-insensitive and accent-insensitive matching is selected with `-c` and `-d`, respectively.[7] The example below is similar to the CQP query `[lemma = "over.+" %c];` but may be considerably faster on a large corpus.

```
$ cwb-lexdecode -f -P lemma -p 'over.+' -c VSS
```

An **entire corpus** or selected attributes from a corpus can be printed in various formats with the `cwb-decode` tool. Note that options and switches must appear *before* the corpus name, and the flags used to select attributes *after* the corpus name. Use `-P` to select p-attributes and `-S` for s-attributes. With the `-s` and `-e` options, a part of the corpus (identified by start and end corpus position) can be printed.

```
$ cwb-decode  -C -s 7299 -e 7303  VSS  -P word -P pos -S s
```

`-C` refers to the compact one-word-per-line format expected by `cwb-encode`. For a full textual copy of a CWB corpus, use `-ALL` to select all positional and structural attributes.

```
$ cwb-decode  -C  VSS  -ALL  > vss-corpus.vrt
```

The resulting file *vss-corpus.vrt* can be re-encoded with `cwb-encode` (using appropriate flags) to give an exact copy of the `VSS` corpus. `-Cx` is almost identical to the compact format, but changes some details in order to generate a well-formed XML document (unless there are overlapping regions or s-attributes with "simple" annotations).[8]

```
$ cwb-decode  -Cx  VSS  -ALL  > vss-corpus.xml
$ xmllint --noout vss-corpus.xml  # not well-formed :-(
```

This output format can reliably be re-encoded when the `-xsB` options are used. Finally, `-X` produces a native XML output format (following a fixed DTD), which can be post-processed and formatted with XSLT stylesheets.

```
$ cwb-decode  -X -s 7299 -e 7303  VSS  -P word -P pos -S s -S np_head
```

[7] Recall that these flags are only guaranteed to work correctly for a corpus in ISO-8859-1 (Latin-1) encoding.

[8] In order to re-create the original input file *vss.vrt* as a well-formed XML document, it would have been necessary to store the full strings of attribute-value pairs from XML start tags by using `-V` flags instead of `-S` in `cwb-encode` attribute declarations (e.g. `-V story:0+num+title`). In the `cwb-decode` call, problematic s-attributes created by auto-splitting of these attribute-value pairs (`story_num`, `story_title`, `s_len`, `np_head`, ...) can then be omitted. The specification `-S story` would print the full attribute-value pairs in `<story>` tags, etc.

Note that the regions of s-attributes are not translated into XML regions. Instead, the start and end tags are represented by special empty `<tag>` elements.

The `cwb-scan-corpus` computes **combinatorial frequency tables** for an encoded corpus. Similar to the `group` command in CQP, it is a faster and more memory-efficient alternative for the extraction of simple structures from large corpora, and is not restricted to singletons and pairs. The output of `cwb-scan-corpus` is an unordered list of n-tuples and their frequencies, which have to be post-processed and sorted with external tools. The simple example below prints the twenty most frequent (`lemma`, `pos`) pairs in the `VSS` corpus, using the `-C` option to filter punctuation and noise from the list of lemmata (note that `-C` applies to *all* selected attributes).

```
$ cwb-scan-corpus -C VSS lemma pos | sort -nr -k 1 | head -20
```

A non-negative **offset** can be added to each field key in order to collect **bigrams**, **trigrams**, etc. The following example derives a simple language model in the form of all sequences of three consecutive part-of-speech tags together with their occurrence counts. Only the twenty most frequent sequences are displayed.

```
$ cwb-scan-corpus VSS pos+0 pos+1 pos+2 | sort -nr -k 1 | head -20
```

For a large corpus such as the `BNC`, the scan results can directly be written to a file with the `-o` switch. If the filename ends in `.gz` (such as the file *language-model.gz* in the example below), the output file is automatically compressed (using `gzip`).

```
$ cwb-scan-corpus -o language-model.gz BNC pos+0 pos+1 pos+2
```

The values of the selected p-attributes can also be filtered with regular expressions. The following command identifies part-of-speech sequences at the end of sentences (indicated by the tag `SENT` = sentence-ending punctuation).

```
$ cwb-scan-corpus VSS pos+0 pos+1 pos+2=/SENT/ | sort -nr -k 1 | head -20
```

Since the third key is used only for filtering, we can suppress it in the output by marking it as a constraint key with the `?` character. Note that it may be necessary to enclose more complex keys (containing shell metacharacters) in single quotes.

```
$ cwb-scan-corpus VSS pos+0 pos+1 ?pos+2=/SENT/ | sort -nr -k 1 | head -20
```

Note that `cwb-scan-corpus` can operate both on p-attributes and on s-attributes with annotated values. To obtain by-story frequency lists for the `VSS` corpus, use the following command:

```
$ cwb-scan-corpus -o freq-by-story.tbl VSS lemma+0 story_title+0
```

As a special case, s-attributes without annotated values can be used to restrict the corpus scan to regions of a particular type. For instance, the constraint key `?footnote` would only scan `<footnote>` regions. Keep in mind that such special constraints must not include a regular expression part.

The final example extracts pairs of adjacent adjectives and nouns from the `VSS` corpus, e.g. as candidate data for adjective-noun collocations. Constraint keys are used to identify adjectives and nouns, and only nouns starting with a vowel are accepted here. Note the `c` and `d` modifiers (case- and diacritics-insensitive matching) on this regular expression.

```
$ cwb-scan-corpus -C VSS lemma+0 ?pos+0=/JJ.*/ lemma+1=/[aeiou].+/cd ?pos+1=/NN.*/
```

Except for the `-C` option, this command line is equivalent to the following CQP commands, but it will execute much faster on a large corpus.

```
> A = [pos = "JJ.*"] [pos = "NN.*" & lemma = "[aeiou].+" %cd];
> group A matchend lemma by match lemma;
```

The `cwb-scan-corpus` command is limited to relatively simple constraints on tokens and it can only match patterns with fixed offsets (but not e.g. determiner and noun separated by an arbitrary number of adjectives). To obtain frequency tables for more complex patterns, use CQP queries in combination with the `tabulate` function. The resulting data tables can be saved to disk files and loaded into a relational database or processed with a software package for statistical analysis.

8 Sentence alignment

An alignment between two parallel corpora (e.g. a collection of source texts and their translations into some other language) can be encoded as a corpus attribute within CWB.

- *Alignment attributes* (a-attributes) are unlike other types of attribute because *alignment presupposes the existence of the source and target corpora.* That is, *first* we need to encode the two corpora independently; *then* we can add the alignment attribute that links them.
- Alignment attributes are usually employed for sentence alignment, and this tutorial will assume throughout that it is sentences that we are aligning.
- However, you can also align at some other level (e.g. clauses or paragraphs or chapters). Aligning regions that are much smaller than a sentence will not be very useful because of the limitations of how CQP deals with a-attributes.
- Only one a-attribute linking any particular pair of corpora can exist.
- There are two ways that a pair of corpora can be aligned.
  - First, the `cwb-align` tool can be used to automatically align the sentences of the two corpora, with its output subsequently encoded as an a-attribute using `cwb-align-encode`.
  - Second, an existing alignment scheme encoded in the corpus markup can be imported as an a-attribute using `cwb-align-import`.

CWB supports many types of alignment link: one to one, many to many, and crossing. However, the regions in the corpora that are the units to be aligned with one another cannot be discontinuous.

8.1 The example corpora

First, let's introduce the data we'll be working with in this part of the tutorial. All the files mentioned here are available as part of the **data package** provided alongside the tutorial document. The corpus we'll use to practice alignemnt consists of a very short excerpt from the novel *The Hound of the Baskervilles* by Arthur Conan Doyle, which we'll call the *Holmes* corpus after the main character. As well as the original English, we have a German translation of the same text. We'll use the CWB labels `HOLMES-EN` for the source corpus and `HOLMES-DE` for the target corpus (i.e. the translation) respectively. Using language codes to distinguish components of a parallel corpus in this way is a useful way to organise labels for aligned corpora in CWB.

Before going further in the tutorial, you should index these two corpora, using the following commands:

```
<p num="3">
<s id="a">
Mr.         NP     Mr.
Sherlock    NP     Sherlock
Holmes      NP     Holmes
[...]
was         VBD    be
seated      VBN    seat
at          IN     at
the         DT     the
breakfast   NN     breakfast
table       NN     table
.           SENT   .
</s>
<s id="b">
I           PP     I
[...]
stood       VBD    stand
upon        IN     upon
the         DT     the
hearth-rug  NN     hearth-rug
and         CC     and
picked      VBD    pick
up          RP     up
the         DT     the
stick       VB     stick
[...]
.           SENT   .
</s>
[... two more sentences ...]
</p>
```

Figure 5: Example from the source corpus (file *holmes_ en.vrt*), with abbreviations

```
$ cwb-encode -d /corpora/data/example -c utf8 -f holmes_en.vrt
             -R /usr/local/share/cwb/registry/holmes-en
             -P pos -P lemma -S s+id -S p+num
$ cwb-encode -d /corpora/data/example -c utf8 -f holmes_de.vrt
             -R /usr/local/share/cwb/registry/holmes-de
             -P pos -P lemma -S s+id -S p+num
```

(you should, of course, amend the `-d` and `-R` options to suit your own setup).

All the example commands given in the following sections are based on these two corpora. They do not include the `-r` option to specify the registry directory location. If you have placed the registry files for the two corpora anywhere other than the default registry, you will need either to add the `-r` option, or else to use the CWB_REGISTRY environment variable.

```
<p num="3">
<s id="a">
Mr.              NN       Mr.
Sherlock         NN       Sherlock
Holmes           NE       Holmes
[...]
saß              VVFIN    sitzen
am               APPRART  an
Frühstückstisch  NN       Frühstückstisch
,                $,     ,
während          KOUS     während
ich              PPER     ich
auf              APPR     auf
dem              ART      die
Kaminvorleger    NN       Kaminvorleger
stand            VVFIN    stehen
und              KON      und
den              ART      die
Spazierstock     NN       Spazierstock
aufhob           VVFIN    aufheben
[...]
</s>
[... three more sentences ...]
</p>
```

Figure 6: Example from the target corpus (file *holmes_de.vrt*), with abbreviations

8.2 Using the sentence aligner

The `cwb-align` program is a very simple text aligner. It can be considered a "fallback" option for sentence alignment, designed to provide basic functionality when nothing better is available. If your corpus is already aligned, it is always better to use that existing alignemnt data. Similarly, if you have a properly-designed and trained aligner for a given language pair, it is always better to use that than to reply on `cwb-align`.

In particular, `cwb-align` will not work well on languages that are unrelated to the extent of sharing little or no vocabulary, as it works by looking for similarities in the words used in the two corpora it analyses.

`cwb-align` makes use of very basic techniques to align units in two parallel corpora by spotting units - assumed to be of about sentence length - that have the same content. It looks for similarities in terms of:

- The length of each corpus segment, measured in characters.
- The presence of shared words across the two corpora (ignoring case and accents).
- The presence of shared letter sequences (for spotting similar but not identical words).
- The presense of words specified as translation equivalents (a file containing the list of word-pairs must be provided to look for these kinds of similarity).

Here is how we might **create an alignment from scratch and then encode it** using the two `HOLMES` corpora, assuming that the `<s>` elements are the units to be aligned.

The most basic use of `cwb-align` would be as follows:

```
$ cwb-align -o holmes.align HOLMES-EN HOLMES-DE s
```

This command has one option and three arguments. The `-o` option simply specifies a filename for the output data. The first and second arguments are the labels of the source corpus and the target corpus respectively. The third argument is the *grid attribute*, that is, the s-attribute used as the alignment grid.

The output file has five columns (see figure 7).

The first line is a header line with the names of the aligned corpora and of the grid attribute.

Each subsequent line specifies a pair of aligned regions:

- The beginning of the region in the source corpus
- The end of the region in the source corpus
- The beginning of the region in the target corpus
- The end of the region in the target corpus
- The type of alignment: 1:1, 2:1, 1:2 or 2:2 indicating one-to-one, two-to-one, etc. etc.
- A number indicating how sure the alignment engine is that this pair of regions really matches (the "quality").

However, it is not normally necessary for a human being to read the file. Usually it would be used only as input data for the next step (see below).

```
TODO insert here what the output looks like
```

Figure 7: Output from the most basic use of the aligner

To check whether the aligner worked correctly, you can view this file interactively using the `cwb-align-show` program. The command to run this program is:

```
$ cwb-align-show holmes.align
```

(you can use the `-w` option for a wider display, if your terminal window is big enough).

Press `Return` to display the next alignment pair, `h` for other key commands, and `q` to exit the viewer.

If your parallel corpus is large, it may be advisable to compress the `.align` file by specifying a filename with extensions `.gz`. All CWB alignment tools handle such compressed files transparently.

8.3 Advanced use of the aligner

It is possible to get improved results from `cwb-align` by making use of different parts of the original corpora, or by tweaking the configuration of the weight it gives to different kinds of comparison.

One tweak we can make is to the p-attribute used by the aligner to measure similarity. The following command, for instance, will used the `lemma` attribute as the "text" of the corpora when comparing their content:

```
$ cwb-align -P lemma -o holmes.align HOLMES-EN HOLMES-DE s
```

The different p-attributes need to have the same name in both source and target corpora for this to work.

There are different reasons why you might use an attribute other than the default `word` for the lexical comparisons. You might choose to use the `lemma` attribute, for instance, if the two languages are closely related but differ in their inflections (in which case the lemmata would be overall more similar to one another, and thus easier to align, than the actual word-tokens). Alternatively, you might choose to align using the `lemma` attribute if you had a bilingual lexicon available which contained lemmata. `cwb-align` is able use such a lexicon if it is available: words which are identified in the lexicon as equivalent will then count as "similar" for alignment purposes even if they are formally nothing alike.

```
be sein
sit sitzen
stand stehen
```

Figure 8: A very short English-German bilingual lexicon file, *lex.txt*

The format of a lexicon file is shown in figure 8. The aligner can be instructed to use it as follows:

```
$ cwb-align -P lemma -o holmes.align HOLMES-EN HOLMES-DE s -W:50:lex.txt
```

The `-W` option is an *aligner configuration flag*, so it goes after the names of the corpora and the grid attribute (in contrast to the *general options*, which precede the names of the corpora.

Note that when the `-W` flag is used, you must specify two things: first the *weight* to be given to words that match when aligning sentences, and then the name of the file containing the pairs equivalents. The weight given in the example above, 50, is equal to the default weight given to an occurence of the exact same word in both languages. This number is one of the paramters that you can change to try to improve the alignment output; see also below. The second thing that must be specified is, of course, the name of the lexicon file.

There are many other parameters that can be tweaked and it may be worth experimenting to see what gives you the best results. We won't cover the details in this tutorial. All are described in full in the `cwb-align` manual file (accessed by the command `man cwb-align` on Unix, provided as a separate file on Windows).

One thing worth noting, however, is that it is possible to use *pre-alignment*. "Pre-alignment" means that some correspondances are known in advance. In a novel, for instance, there may be chapter boundaries which match across translations, and we can say for certain that a sentence in chapter 1 in language A will not be aligned with a sentence in any other chapter than 1 in language B. This makes the aligner's task easier.

If the indexed corpora contain such pre-alignment information encoded as an s-attribute, then the aligner can be instructed to use it.

In the `HOLMES` corpora there exist paragraphs (s-attribute `p`). Let us assume that these paragraphs are pre-aligned: we know that a given paragrpah in `HOLMES-EN` matches one and only one paragraph in `HOLMES-DE`, and that these links are known; it is only the alignment of sentences within each paragraph pair that needs to be found out.

In this case we can add either the `-S` or `-V` option to `cwb-align`.

If we specify pre-alignment with `-S`, then the aligner assumes that the source and target corpora have the same number of paragraphs, and that the first paragraph in the source (`HOLMES-EN`) corresponds

to the first paragraph in the target (`HOLMES-DE`), the second to the second, and so on. This would be done as follows:

```
$ cwb-align -S p -o holmes.align HOLMES-EN HOLMES-DE s
```

Alternatively we can use `-V`. In this case, paragraphs will not be matched up by order - they are matched up by the value of the s-attribute. Since the *Holmes* corpora input data have *num* as an annotation, there is an s-attribute `p_num` which has values and can be used in this way. This would be done as follows:

```
$ cwb-align -V p_num -o holmes.align HOLMES-EN HOLMES-DE s
```

In this case, the order of the paragraphs does not matter: the aligner will always try to match sentences in paragraph 3 in one corpus to sentences in paragraph 3 in the other corpus.

Using pre-alignment improves the output, because fewer possibilities have to be checked for the alignment of each sentence.

8.4 Encoding the aligner's output

An alignment attribute is added to an existing CWB corpus, which must be the *source* corpus of the alignment (not the target). There are two steps in this process.

The first step is to declare the new alignment attribute in the source corpus's registry file.

So, find the `holmes-en` file in the registry directory, and edit it to add the following line:

```
ALIGNED holmes-de
```

(note the use of the lowercase spelling of the attribute name!)

This declares an a-attribute linking this corpus to the `HOLMES-DE` corpus. An a-attribute has the same name as the target corpus.

If you've got the CWB/Perl tools installed, you can use the `cwb-regedit` to make this change, rather than manually editing the registry. The command would in this case be as follows:

```
$ cwb-regedit HOLMES-EN :add :a holmes-de
```

Once the registry file has been updated, the second and final step is to encode the alignment attribute:

```
$ cwb-align-encode -D holmes.align
```

(note that this command will run very fast and will print no output if everything has gone well).

There is only one argument to the `cwb-align-encode` program: the name of the text file containing the alignment data. It is not necessary to name either of the corpora, because the `holmes.align` file contains both names.

It is, however, always necessary to state where you want the encoded files to be placed. The recommended way to do this is the method shown above: with the `-D` option. This puts the a-attribute's data files in the same directory used for the corpus's other attributes (specified in the registry file).

Alternatively, you can specify a different location with the `-d` option.

Once encoding is complete, it's safe to delete the `holmes.align` file.

This procedure only creates an a-attribute in `HOLMES-EN`, linking it to `HOLMES-DE`. If you *also* want an a-attribute in `HOLMES-DE` linking it to `HOLMES-DE`, you must repeat the whole procedure with the source and target corpora switched. Alternatively, you can run `cwb-align-encode` with the option `-R`, provided that there are no crossing beads in your alignment.

That is, first run

```
$ cwb-regedit HOLMES-DE :add :a holmes-en
```

and then add the a-attribute data to `HOLMES-DE`. It is possible to use the same `holmes.align` file to do this by telling `cwb-align-encode` to switch around the source and target corpora that it finds in that file. The option for this is `-R` ("reverse mode"), as shown here:

```
$ cwb-align-encode -D -R holmes.align
```

8.5 Importing a pre-existing alignment

It may be that your corpora have already been aligned, either manually or using a better aligner than `cwb-align`. In this case, you can create an a-attribute by *importing* such existing alignment information with `cwb-align-import`. Note that `cwb-align-import` is *not* part of the main CWB core, but is instead one of the CWB/Perl tools.

The procedure to **import an alignment from existing information** is as follows.

First, you must encode your information into an **alignment beads file**. An *alignment bead* is one single point of alignemnt between the source and target corpora. An *alignment beads file* is a file defining a series of beads, plus some header information.

The header line of a beads file has four items, separated by tabs:

- The CQP ID of the source corpus
- The CQP ID of the target corpus
- The ID of s-attribute encoding the regions to be aligned (i.e. the "grid", as explained above)
- A "key pattern".

The *key pattern* specifies how ID codes for regions in the beads file relate to ID codes for regions in your indexed corpora. Let's assume here that our beadfile will contain IDs that have exactly the same IDs as in our corpora. In *Holmes*, we are aligning sentences that have single letter ID codes in an s-attribute called `s_id`. To express this as a key pattern, you simply put "id" within curly brackets. So the header line would be as follows:

```
HOLMES-EN    HOLMES-DE    s    {id}
```

(For examples of more complex key patterns, see **man cwb-align-import**.)

Lines after the first line must each contain a single alignment bead. A *bead* consists of two columns, separated by a tab. Each column contains one or more space-separated IDs (in our example, from

the `s_id` attribute) to be treated as aligned to one another. The IDs in the first column relate to the source corpus and the IDs in the second column relate to the target corpus.

You can see from the input files for the *Holmes* data that the two sentences in the English version, **a** and **b**, correspond to a single sentence in the German version, with the ID **a**. So our first alignment bead should state that **a** and **b** in the English text (Column 1) correspond to **a** in the German text (Column 2). This would look like this:

```
a b    a
```

Second, having created or generated a beadfile, you can import it, creating the a-attribute, using the following command:

```
$ cwb-align-import -p beadfile.txt
```

Note that it is not necessary to specify which corpora are in use - this information is on the header line of the beads file.

The option in use here, `-p`, is short for *prune*. Using it means that if any region IDs are used in the beads file which do not actually occur in both corpora, that bead is just ignored. Without this option, `cwb-align-import` will abort with an error message if it encounters any bad IDs.

Another flag that is often useful is `-i`. This *inverts* the source corpus and target corpus from what is declared in the beads file. It means that you can create two a-attributes, one going each way, from the same beads file. For instance, the command above creates an a-attribute in `HOLMES-EN`, pointing at `HOLMES-DE`. For an a-attribute in `HOLMES-DE`, pointing at `HOLMES-EN`, use this command:

```
$ cwb-align-import -i -p beadfile.txt
```

A Appendix: Registry file format

The following is a sample registry file created by **cwb-encode**. The **cwb-regedit** also creates registry files in this format.

```
##
## registry entry for corpus BNCSAMPLER
##

# long descriptive name for the corpus
NAME ""
# corpus ID (must be lowercase in registry!)
ID   bncsampler
# path to binary data files
HOME /home/Corpora/data/bncsampler
# optional info file (displayed by "info;" command in CQP)
INFO /home/Corpora//bncsampler/.info

# corpus properties provide additional information about the corpus:
##:: charset  = "utf8" # change if your corpus uses different charset
##:: language = "??"     # insert ISO code for language (de, en, fr, ...)
```

```
##
## p-attributes (token annotations)
##

ATTRIBUTE word
ATTRIBUTE pos
ATTRIBUTE hw
ATTRIBUTE semtag
ATTRIBUTE class
ATTRIBUTE lemma


##
## s-attributes (structural markup)
##

# <text id=".."> ... </text>
# (no recursive embedding allowed)
STRUCTURE text
STRUCTURE text_id              # [annotations]

# <s> ... </s>
STRUCTURE s


# Yours sincerely, the Encode tool.
```

CWB traditionally had a more flexible registry file format (which is still accepted for backward compatibility), which could contain a variety of other declarations. The standard format for new corpora, however, is as given above; we recommend that you stick to this format, since it is in fact enforced by the CWB/Perl scripts.

Finally, it is worth noting that directory and file paths in `HOME` and `INFO` entries have to be *double*-quoted if they contain blanks or other non-standard characters (ASCII letters, digits, `-`, `_`, `/` and `.` are ok, as long as the path does not begin with `.`). In a double-quoted path, `"` must be escaped as `\"` and the backslash `\` as `\\`. If you use `cwb-encode` and `cwb-regedit`, they should always create valid entries, with quotes added when necessary.

B Appendix: Limitations

This section documents some technical limitations of CWB. Most of these are due to the design of the index file format (notably the pervasive use of signed 32-bit integer values), but some additional restrictions are imposed by implementation decisions.

- The maximum corpus size is 2,147,483,647 tokens (the largest value that can be stored in a signed 32-bit integer). In the CWB source code, it is represented by the macro `CL_MAX_CORPUS_SIZE`.
- The maximum size of a p-attribute lexicon is 2,147,483,647 bytes (or 2 GiB), due to the use of signed 32-bit integer offsets into the `.lexicon` file.[9]

[9] Lexicon size refers to the sum of the byte lengths of all annotation strings in the lexicon, including NUL terminators.

- The same 2 GiB limit applies to the "lexicon" storing all distinct annotation strings of an s-attribute with values (the `.avs` file).
- The maximum length of an annotation string is 4096 bytes, including the NUL terminator. Note that in UTF-8 encoding, a single character may occupy two or more bytes, so the maximal character length of an annotation string may be smaller.

  The length limit is determined by the macro `CL_MAX_LINE_LENGTH` in the CWB source code and can be increased if absolutely necessary. This is strongly discouraged, as it may create index files that are not compatible with standard versions of CWB.
- The maximum length of a filename is determined by the macro `CL_MAX_FILENAME_LENGTH` in the CWB sourcecode. Its default value is currently 1024 bytes.
- The maximum length of an input line in the `.vrt` format is currently 65,536 bytes. It is determined by the macro `MAX_INPUT_LINE_LENGTH` in the **cwb-encode** source code.

