N-GRAM PROCESSOR 0.6
A SHORT USER GUIDE

Andreas Buerki
buerkiA@cardiff.ac.uk

CONTENTS

1 INTRODUCTION

This brief guide introduces the reader to what the N-Gram Processor (NGP) is and how it is used. The NGP is an open-source, free software package which produces lists of word n-grams and their frequencies from input text files. Typical users will be linguists working with corpus linguistic methods. Its feature set includes the following:

- creation of word n-gram lists out of input text, incl. frequencies
- listing of document counts
- menu-based interface (new for version 0.6)
- unicode support
- support for processing of large corpora (> 10 million words, given hardware capacity)
- support for processing of annotated corpora

The main feature of the NGP is the production of word n-gram lists out of input text, where word n-grams are word sequences of length n, so a 2-gram is a sequence of two words, a 3-gram one of three, etc. Lists of any length n-grams (within reason) can be produced but each list will only contain n-grams of one particular n (for software that will consolidate lists of n-grams of different n, see the open-source SubString package (`http://buerki.github.com/SubString/`, cf. also O'Donnell, 2011). The way NGP produces n-gram lists is illustrated in table 1 where 2-grams 1 to 7 (upper part) and 3-grams 1 to 6 (lower part) are produced from an input text. N-grams are created by moving a window of size n progressively through the text and listing resulting n-grams. Each n-gram is then gathered into a list and their frequencies indicated (in table 1, the frequencies would all be 1 since each n-gram occurs only once).

One of the principal uses of n-gram lists produced by the NGP is in the context of the extraction of formulaic sequences, multi-word expressions and the like out of corpus data. Here, n-gram lists serve as raw material to an identification of relevant items.

		This	is	an	example	–	an	idealised	example
2-grams	1	This	is						
	2		is	an					
	3			an	example				
	4				example	–			
	5					–	an		
	6						an	idealised	
	7							idealised	example
3-grams	1	This	is	an					
	2		is	an	example				
	3			an	example	–			
	4				example	–	an		
	5					–	an	idealised	
	6						an	idealised	example

Table 1: Splitting of text into n-grams.

2 SOME BACKGROUND

I put the NGP together while working on my PhD thesis to help me study diachronic change in common expressions (i.e. formulaic sequences) using corpus data. The NGP shares the core of its code base with the N-Gram Statistics Package (NSP), specifically with versions 1.09 (by Ted Pedersen, Satanjeev Banerjee and collaborators, cf. Banerjee and Pederson, 2003) and 1.10 (a re-write of the NSP by Bjoern Wilmsmann, cf. Wilmsmann, 2007). The two packages are, however, entirely separate and do not interfere with installations of each other. For the benefit of those already familiar with the NSP, the main differences between the two software packages lie in the focus of the NSP on the calculation of statistical measures of association of listed n-grams (hence the 'statistics' in the name), whereas the NGP focuses on the ability to process much larger amounts of data and on the processing of multilingual data, including corpus material written in non-Roman script. Additionally, it makes it possible to track document frequency (i.e. the number of documents in which an n-gram occurs) in addition to global frequency which can be of great importance for assessing the distribution of particular n-grams. The NGP does not include a statistics module (though it is possible to use the NSP statistics module on output data if a certain combination of options is used, see section 6).

3 INSTALLATION

Compatible Operating Systems

The NGP is written in Perl with additions in the bash shell scripting language. As such, it can be run on wide range of operating systems, including Linux, OS X and under the Cygwin environment on Windows. All recent versions of these systems include perl version 5.12 and bash version 3.2 or later which is minimally required to run the NGP. The NGP was tested and confirmed working on OS X version 10.10 and 10.11, Xubuntu 14.04 and Cygwin on Windows 7 and Windows 8.

Double-clickable installers are provided for OS X, Xubuntu-Linux and the Cygwin environment under Windows. For other environments, please follow the manual installation instructions further down.

OS X and Xubuntu

Inside the `ngramprocessor_0.0` directory, double-click on `Xubuntu_installer` (for Xubuntu) or the `OSX_installer` (for OS X). Follow the instructions of the installer. OS X might prompt users to install the command line tools – this is a free download from Apple and is needed to install the NGP.

Cygwin / Windows

Under Windows, before the NGP is installed, the Cygwin environment needs to be installed. This is accomplished by following the steps in the box below:

1. Download and then open the application `setup-x86.exe`, available free of charge from `http://cygwin.com/setup-x86.exe`
2. Follow the on-screen instructions. The default installation settings can be used, except in the following:
   (a) On the 'Choose Installation Directory'-screen, the 'root directory' should be either `C:\cygwin64` (if 64 bit version) or `C:\cygwin` otherwise. This should be the default.
   (b) On the 'Select Packages' screen, the following packages need to be installed in addition to the default packages:

   'bc' from the 'maths' category
   'make' from the 'devel' category
   'makemaker' from the 'perl' category
   'diffutils', 'ncurses' and 'cygutils-extra' from the 'utils' category

   The easiest way to do this is to enter these names (one after another) in the search box, then click the '+' next to 'Utils' (or 'Math' for `bc`) and click on the circling arrows (cf. figure 2). For all other prompts, just click on 'next' or 'okay'. There is no need to tick to create a desktop shortcut or startmenu item on the last screen.
3. Once all configuration options are set, the installer then proceeds to install all necessary components of the Cygwin environment.

Once the Cygwin environment is installed, the NGP is installed in four steps:

1. Open the `ngramprocessor_X.X` folder.
2. Double-click on the `Cygwin_installer` or `Cygwin64_installer` icon (depending on whether or not the 64-bit version of Cygwin was installed. Try both if one doesn't work).
3. The installer window will open, displaying processing information and finally asking if a shortcut should be placed on the desktop. Answer by pressing ENTER.

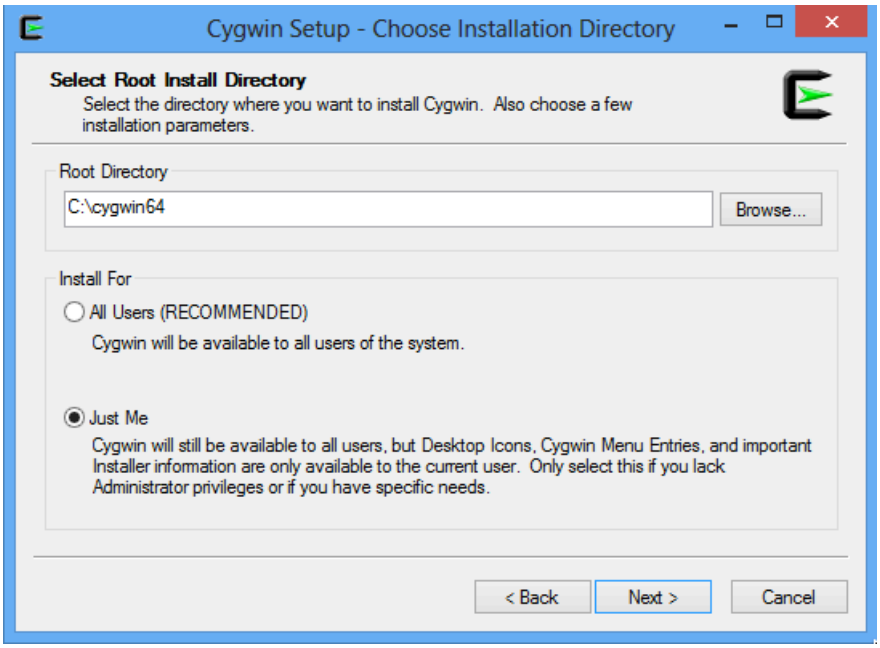

Figure 1: The 'Choose Installation Directory' screen of the Cygwin installer

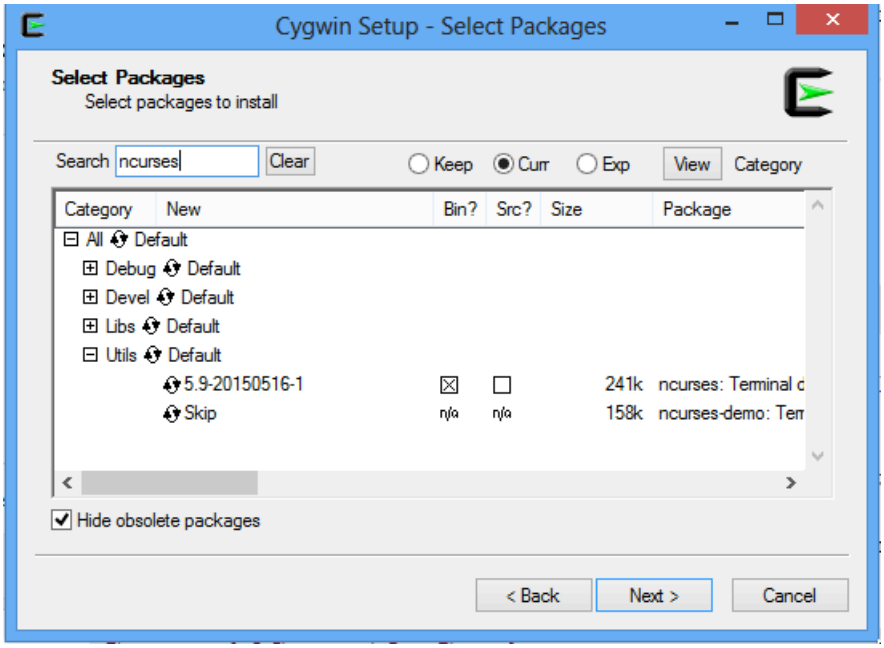

Figure 2: The 'Select Packages' screen of the Cygwin installer

4. A black NGP-icon should appear on the desktop (and also inside the `ngramprocessor_X.X` folder). The NGP is launched by double-clicking on this icon. The icon can be placed anywhere.

The NGP displays and outputs unicode and will therefore work most reliably if the environment is set to use and display data in UTF-8.

Manual installation / other flavours of Linux

1. open a Terminal window

   OS X: in Applications/Utilities

   Xubuntu Linux: via menu Applications>Accessories>Terminal

   Cygwin: via the link on the Windows desktop to Cygwin Terminal

2. drop the `install.sh` script (located inside the `ngramprocessor_X.X` directory) onto the terminal window and press ENTER. This should start the installation process.

If an entirely manual installation is necessary, type the following commands into a terminal window while in the `ngramprocessor_X.X` directory:

```
perl Makefile.PL
make
make test
make install
```

The last command requires administrative privileges, so it might need to be run as `sudo make install`, for example on OS X. This installs the files in the standard locations.

4 THE MENU-BASED INTERFACE

To use the NGP, double-click its icon on the desktop or open a terminal window, type `NGP.sh` and press ENTER. The welcome screen will appear as shown in figure 3 (tt might be necessary to adjust the size of the window to accommodate the whole text). Typing `Y` and ENTER will apply the listed parameters to the extraction. Replying `N` will give the user the chance to adjust all of the listed parameters for a customised extraction. For parameter c., either no stoplist at all can be used or a correctly formatted stoplist can be provided (cf. figure 6, on p. 12). Similarly, for option e., a specially formatted plain text file giving a token definition can be provided for use. This text file must only contain the token definition, which starts with a slash, then listing the tokens, each separated by a pipe (vertical bar symbol) and then ending in a slash, for example /\w+|-|'|&|§|%|\/|\+|ß/. Characters that have special meaning in Perl need escaping to get their literal meaning. \w+ refers to all alphanumeric characters, including underscores and any of the characters following the \w+ symbol, while being treated as tokens, are treated as *separate* from the alphanumeric tokens, such that *bus-stop* is a 3-gram (with *bus*, -, and *stop* representing separate constituents). For more detail on adjusted stoplists and token definitions, see p. 11 ff.

On the next screen, the user is asked to drag a folder with text files (or a text file) into the window. The folder must contain all and only those text files from which n-grams are to be extracted and they must be in plain text format (for exceptions, see p. 12, below).

```
N-GRAM PROCESSOR
version 0.6


Would you like to use the following standard extraction parameters?
a.  n-grams of length 2 to 7 orthographic words
b.  global frequencies and frequencies per document
c.  additive stoplist of top 200 English words
d.  no n-grams across sentence boundaries
e.  alphanumeric characters and the following count as tokens:  -'&§%/+°ß
f.  window size = n-gram size
g.  minimum frequency:  2 per document (or globally if no document
frequencies derived)


(y) yes (n) no (x) exit
>
```

Figure 3: The welcome screen

Warning about spaces in file and folder names
Some operating systems allow spaces in file and folder names. These tend to cause errors in perl programmes and it is therefore best to make sure that the folder dragged into the window and any parent folders do NOT contain spaces. If necessary, the names of files and folders must be changed.

For illustration, the folder called `test_corpus`, found in the test directory inside the `ngramprocessor_X.X` directory, can be used. The NGP then extracts n-grams from the text files and places them into a folder in the current directory called `N-gram_lists`. Note that no list is produced for n-gram sizes where there are no n-grams above minimum frequency. The output lists are plain text files and can be opened in any application that reads plain text files. An example of an output file is shown in figure 5, below: the first line gives the total global frequency of all n-grams on the list, then each n-gram is listed, followed by its global frequency, followed by the number of documents in which it occurred. On Windows, the '<>' sign is used to separate the words in an n-gram, instead of the middle dot (·).

5 THE COMMAND LINE INTERFACE

For more flexibility and finer control of extraction parameters, the various modules of the NGP can be accessed via the command line without the textual menus shown in the previous section. This will be useful for advanced users. *The following examples assume that the current working directory is the `test` directory inside the `ngramprocessor_X.X` directory.*

The programme list.pl

To create an n-gram list, `list.pl` is used. It requires the following pattern

- `list.pl [OPTIONS] OUTFILE IN-DIR/FILE+`

where OUTFILE is the name of the list to be produced, IN-DIR/FILE+ is either one or several input files or a directory with input files in it. The following command will produce an n-gram list called out1.txt out of the input file text1.txt:

```
list.pl out1.txt text1.txt
```

Examining the output list (`out1.txt`), the beginning of which is shown in figure 4, a number of things are noticed. On the first line, the total number of n-gram tokens is shown. This is followed by the list of n-gram types and their frequencies, one per line. By default, 2-gram lists are produced and by default the words in n-grams are separated by an interpunct (or middle dot). The number after the final interpunct is the frequency of the n-gram. Although invisible, a single space follows the frequency on each line. Also by default, n-grams across line breaks (i.e. across the newline character) are excluded. If the -l option is passed, n-grams spanning line breaks are included in the list as well.

```
225
the·ngp·5
ngp·is·4
and·the·3
n·--·3
of·the·3
as·well·2
The·N·2
...
```

Figure 4: The first few lines of out1.txt

To see a bit more of what is going on, the `-v` option can be used. To specify the *n* of the n-grams, `-n` is used and we can vary the separator using `-p`[1]. The full range of available options to the programmes included in the NGP is listed in the appendix below and is available for each programme by calling the -h (help) option.

The following command produces a 6-grams list out of `text1.txt` using <> as separator:

```
list.pl -v -n 6 -p '<>' out2.txt text1.txt
```

Instead of passing single input files, we can also pass directories with input files in them. The following will produce a single output list of 3-grams (out3.txt) from both text files in the directory `files`.

[1] Although the separator can be varied freely in `list.pl`, two things should be kept in mind when choosing a separator symbol: 1) it must be a symbol that is NOT found at all in the input texts (otherwise confusion will result). If necessary, input files might need to be checked and offending characters removed or replaced before processing. 2) `unify.pl` automatically recognizes two types of separators: the interpunct (·) and the diamond (<>), and so it is easiest to use one of those, although `unify.pl` allows any separator to be specified via its `-p` option.

```
list.pl -v -n 3 out3.txt files
```

If only n-grams of a certain frequency are to be included in the list, the -f option to `list.pl` can be used to specify the minimal frequency an n-gram has to occur with in order to be included in the list. The following command will only include n-grams that occur with a minimal frequency of 2. The effect is merely to exclude the lower frequency n-grams from the list; the total number of n-gram tokens displayed at the top of the list remains the same.

```
list.pl -vf 2 -n 3 out4.txt files
```

The programme unify.pl

So far, only global frequencies were listed for each n-gram, that is, the frequencies across *all* input files. If the number of documents in which an n-gram occurs should additionally be listed, n-gram lists first need to be produced for each document, and then combined using `unify.pl`.

- `unify.pl [OPTIONS] IN-DIR/FILE+`

Using the commands below, first two 3-gram lists are created, one each of `text1.txt` and `text2.txt`. Then they are unified using `unify.pl`.

```
list.pl -v -n 3 out5.txt text1.txt
list.pl -v -n 3 out6.txt text2.txt
unify.pl -vd outcombined.txt out5.txt out6.txt
```

If the output file (outcombined.txt) is now examined, the document count appears two spaces after the global frequency of each n-gram. The -d option passed to `unify.pl` was responsible for adding the document count – if it is left out, an output file that is identical to `out3.txt`, above, should be the result:

```
unify.pl -v out3bar.txt out5.txt out6.txt
```

In actuality, n-grams in `out3.txt` and `out3bar.txt` are listed in a slightly different order, but n-grams, frequencies and total n-gram token numbers are identical.

multi-list.sh, split-unify.sh and workflows for large corpora

When dealing with large amounts of corpus data from which n-gram lists with global and document frequencies need to be derived, it is not feasible to go about the task by running list.pl an unify.pl in the manner shown above. Depending on how the corpus is structured, the first step in a workflow deriving n-gram lists from the entire corpus using the NGP is to divide the corpus into individual constituent documents. Unless any existing corpus annotations must be preserved in n-grams extracted (see below for what to do in such cases), any markup or annotation (except for line breaks which are relevant to n-gram creation)[2] should be removed.

[2] See -n or --newline option to `list.pl` and `multi-list.sh`

In a second step, n-gram lists are then prepared for each corpus document. Unless individual corpus documents are extremely large (nearing five million words of text), there should be no issues with RAM limits at this stage. To facilitate this step, the shell script `multi-list.sh` can be used to automatically run list.pl on all corpus documents in a given directory (note that although list.pl also accepts directories with multiple files as input, it creates a total of *one* output list, rather than one output n-gram list for *each* of the corpus documents). `multi-list.sh` is nothing more than a wrapper that calls on `list.pl` to produce n-gram lists for each of the input files. The way `multi-list.sh` is called closely parallels the way `list.pl` itself is called (it also has most of the same options available):

- `multi-list.sh [OPTIONS] OUT-DIRECTORY IN-DIRECTORY`

OUT-DIRECTORY is the directory into which the output is placed, IN-DIRECTORY is a directory with files to be processed (unlike `list.pl`, `multi-list.sh` does not accept a list of files to be processed – only a directory can be specified as input). `multi-list.sh` places a new directory, containing the output list(s), into the OUT-DIRECTORY.

The following command places a new directory in the current working directory (represented by the dot) and fills it with one 3-gram list for each of the files present in the directory `files`. The exact name of the output directory depends on options passed to `multi-list.sh`, but it will always contain within it the *n* of the n-gram lists it contains, followed by the word 'comp' for comprehensive (if a single list is contained) or 'per_doc' (if one list per input document is contained within it).

```
multi-list.sh -vdn 3 .  files
```

The -d option makes sure that one output list is produced for *each* input document. If the -d option is not invoked, `multi-list.sh` operates much like `list.pl` in that it produces a single n-gram list from all the input files. As before, -v and -n 3 invoke verbose processing and the production of 3-grams (rather than default 2-grams) respectively.

The next step is to combine all those single lists (depending on corpus size, this may be thousands of lists, though in our test data it is only two lists) into one overall list showing both the global frequency and the document frequency for each n-gram. This could of course be accomplished using `unify.pl` as shown above. However, here the RAM bottleneck is encountered: the combination of thousands of n-gram lists in memory fills a very large amount of RAM space (a multiple of the size of the files to be processed). `split-unify.sh` seeks to address this problem by first splitting lists to be combined, and then combining each section of each list before aggregating the combined sections into a full list.[3] The splitting is done in such a way that n-grams which potentially need combining are found in the same section of all input lists, whereas n-grams that will not need combining are in different sections of the input lists. For example, sections A may contain all n-grams starting with the letter *A* in all the source lists, whereas sections B may contain all the n-grams starting with the letter *B* in all the source lists. No n-gram starting with *A* will ever need combining with one starting with *B* (or rather their frequencies will never need combining), but among those n-grams in A-sections there will be many that are identical across A-sections of the various lists to be combined and will therefore need their frequencies summed while

[3] This bottleneck is of course also encountered if we were to produce just one n-gram list from thousands of documents or from one very large document using `list.pl` or `multi-list.sh`. This is why it may be a good idea to have `multi-list.sh` produce lists for each document first and combine those lists into a single list subsequently *even if* no document count is required.

524		
n·-·grams·	5	2
the·ngp·is·	4	2
of·the·ngp·	4	2
of·the·nsp·	3	1
to·the·ngp·	2	2
of·n·-·	2	2
n·-·gram·	2	2
grams·and·their·	2	2
The·N·-·	2	1
N·-·Gram·	2	1
...	...	...

Figure 5: The first few lines of 3.per_doc.lst

appearing as just one entry (one type) on the combined list. I refer to this type of splitting as alphabet splitting.[4] `split-unify.sh` is called in the following manner:

- `split-unify.sh [OPTIONS] IN-DIRECTORY`

To combine the n-gram lists in the directory that was produced by the command we called earlier, the following command is used:

```
split-unify.sh -vd 3.per_doc
```

After this command has run, the directory `3.per_doc` will be prefixed with `indiv_lists_`, showing that it only contains individual un-unified lists and the unified list (named after the name of the input directory + the extension .lst, which can be changed to .txt if necessary.) will appear in the current working directory. The format of the output list, as shown in figure 5, is slightly different from the output in figure 4: frequencies (global n-gram frequencies and document count) are each preceded by a tab for easier reading (this sort of tidying of the output list can be suppressed by passing the -u option to `split-unify.sh`). The -d option ensures that document counts are included. If additionally the -i option is invoked, the directory with individual input lists will be deleted and only the unified output list in the current working directory remains (which can save a lot of disk space). By default, `split-unify.sh` employs a 47-way alphabet split. A finer splitting into 84 sections (and therefore a further reduction in memory use) can be achieved by invoking the -b option (for big). Since splitting will take some time and requires a lot of read/write activity, the alphabet splitting should only be invoked where necessary. It can be turned off entirely by passing the -s (for small) option. The -m option (for minimum memory) can be invoked to reduce RAM usage to a minimum. It uses an alternative algorithm which assembles the different lists and consolidates them line-by-line, thus reducing RAM requirements drastically, while taking more processing power and typically significantly more time. Table 2 shows the processing modes available in `split-unify.sh`, their function and use. The

[4] Despite the NGP's aim to be multi-language aware, `split-unify.sh` is currently only operating alphabet-splitting for input using Roman characters. Input documents with text written in other character sets do not currently benefit from alphabet-splitting, although other functions are unaffected.

choice of the ideal mode of operation depends on the amounts of data to be processed and the hardware (both RAM and processor speed) used. Generally speaking, if 5 million words or fewer are processed, the -s option will usually be the best choice. If, during processing, the available RAM is exhausted and the operating system starts writing page-outs to disk, the programme becomes inoperable and will stall. Most operating systems have a way of monitoring RAM usage and this can be used to get an idea of how much memory is being used for a particular run. If this happens, either the -b option must be used, or, if memory requirements are still too high, the -m option should be invoked.

Option	Processing mode	Use
-s	(small) no split, employing `unify.pl` directly to process data	Suitable for smaller data-sets (up to around 5 to 10 million words, depending on hardware), necessary if statistical measures of association are to be computed subsequently using the Ngram Statistics Package
none	47-way split; combination with `unify.pl`	Suitable if RAM usage needs to be reduced; not suitable for processing data in non-latin scripts
-b	(big) 85-way split; combination with `unify.pl`	Further reduced RAM requirements, slightly increased processing demands; not suitable for non-latin scripts
-m	(minimal memory) alternative algorithm without calling `unify.pl`	Minimal RAM requirements, increased processing demands. This will take significantly longer to process; suitable for data in any script

Table 2: Processing modes available in split-unify.sh.

In this manner, `split-unify.sh` can reduce memory requirements notably and thus facilitate the processing of larger amounts of data if necessary.[5] The auxiliary scripts `multi-list.sh` and `split-unify.sh` therefore aid the processing of large amounts of data and facilitate the handling of a workflow involving a large number of documents.

Token definitions and Stoplists

When producing word n-grams out of text, a decision needs to be taken as to which characters should be part of n-grams, and which characters should not. The former are defined as tokens, the latter as space. When setting up a token definition, the set of characters that should be treated as part of n-grams are defined, the remaining characters, which commonly includes the space character and tabs but can also include punctuation marks and other special characters, are automatically treated as space. The default token definition used by NGP is displayed by calling -t show (or --token show) in either `list.pl` or `multi-list.sh`:

```
list.pl -t show
multi-list.sh -t show
```

Token definitions are presented in the format of Perl regular expressions. To set a different token definition, a text file containing token definitions needs to be prepared and passed to either `list.pl` or `multi-list.sh` using -t FILE, where FILE is (the path to) a text file

[5] The smallest amount of memory is used if `split-unify.sh` is operated using the -m option, if texts without annotation are processed (see below) and a one-character n-gram separator is used (both '<>' and '·' take more memory because they are binary symbols).

containing the token definition. The token definition should follow the format of starting with a slash, then listing the tokens, each separated by a pipe (vertical bar symbol) and then ending in a slash, for example /\w+|-|'|&|§|%|\/|\+|ß/. Characters that have special meaning in Perl need escaping to get their literal meaning. \w+ refers to all alphanumeric characters, including underscores and any of the characters following the \w+ symbol, while being treated as tokens, are treated as *separate* from the alphanumeric tokens, such that *bus-stop* is a 3-gram (with *bus*, -, and *stop* representing separate constituents).

The NGP features the option of specifying a stop list of words that have the effect of suppressing the listing of certain n-grams. There are two modes: in ABSOLUTE mode, all n-grams are excluded that contain a word from the stop list. In ADDITIVE mode, only n-grams are excluded that are composed entirely of stop-listed words. A stop list can be supplied to either `list.pl` or `multi-list.sh` by passing the option -o and the stop list (here, the example stop list in the test directory is used):

```
list.pl -o stop_en out7.lst files
multi-list.sh -o stop_en .  files
```

Stop list files need to be formatted such that the mode is indicated in the first line (either as @stop.mode=ABSOLUTE or @stop.mode=ADDITIVE), and stop words then listed, one per line, between /^ and $/. An example is shown in figure 6.

```
@stop.mode=ADDITIVE
/^the$/
/^of$/
/^to$/
/^and$/
/^a$/
/^in$/
...
```

Figure 6: The first few lines of the stop_en stop list

Working with annotated corpora

It may be necessary or desirable to preserve certain corpus annotations (such as parts of speech tags or a morphological information) in the n-gram lists produced by the NGP. Generally, since this will increase memory requirements, annotation would ideally be as economic as possible. In order to carry annotation over from the source text to the n-gram lists, it needs to be associated with word tokens. Since only strings of alphanumeric characters are treated as uninterrupted n-gram constituents (see discussion of token definition, above), the underscore must be used to link annotation to the constituent as shown. Input files with annotation attached in this way can be run through `list.pl` or `multi-list.sh` in the manner described above. The file `text1_annot.txt` in the `test` directory is provided as an example of an annotated input text. It contains part of speech tags, attached to word tokens with underscores. Processing in the usual way with `list.pl` yields an n-gram list in which annotation is preserved as shown in figure 7.

```
225
the_DT·ngp_NN·is_VBZ·3
of_IN·the_DT·ngp_NN·3
The_DT·N·-·2
-·Gram_NP·Processor_NP·2
N·-·Gram_NP·2
n·-·grams_NNS·2
the_DT·Ngram_NP·Statistics_NPS·1
methods_NNS·but_CC·others_NNS·1
is_VBZ·structured_VBN·as_RB·1
...
```

Figure 7: The first few lines of a 3-gram list with part-of-speech annotation preserved.

6 COMPATIBILITY WITH THE NGRAM STATISTICS PACKAGE

As mentioned above, the NGP does not include a statistics module and does not therefore produce the type of frequency information that allows the computation of contingency tables such as are required for the calculation of statistical measures of word association like the MI-score or log likelihood scores. Generally, if those are required, use of the NSP rather than the NGP may be preferable. However, it is possible to pass parameters to `list.pl`, `multi-list.sh`, `unify.pl` and `split-unify.sh` in a manner that allows output n-gram lists to be processed subsequently, using the NSP's `statistic.pl` programme, to calculate statistical association measures. Naturally this requires the installation of the NSP alongside the NGP (since they are separate packages, they can both be installed on a single system). For compatible lists to be created, `list.pl`, `multi-list.sh` must be passed the -a option to calculate the additional frequency numbers and the -s option must be passed to `split-unify.sh`. Document frequencies must *not* be included. In addition, the NSP's default separator symbol '<>' must be specified as the separator (using `list.pl` and `multi-list.sh`'s -p option). It should then be possible to use the lists so produced as input to the NSP's `statistic.pl` programme. The following command, for example, will produce a list that should be processable by the NSP's `statistic.pl`:

```
multi-list.sh -vap '<>' .  files
```

Output files for all the example commands used in this manual are provided in the goldstandard_lists directory inside the test directory. These can be used to verify operation of the NGP-installation. Differences are most likely to arise if different encodings are set in the environment. The gold standard lists were produced using the locale en_GB.UTF-8.

7 COMMENTS, LICENSE AND DISCLAIMER

The NGP is licensed under the GNU General Public license: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The NGP is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You

should have received a copy of the GNU General Public License along with the NGP. If not, see http://www.gnu.org/licenses/.

Finally, the NGP repository currently lives at http://buerki.github.io/ngramprocessor/, where there is the possibility to get in touch, report issues and sign up to be notified of updates.

This appendix first gives a synopsis of the programmes included in the NGP and then lists and explains the options available in each. A summary of this information is available also by passing the -h option to the respective programme, or, for Perl programmes, by calling up the manual (e.g. by calling `man list.pl` or `perldoc unify.pl`).

Synopsis

- `list.pl [OPTIONS] OUT-FILE {IN-FILE+|IN-DIRECTORY}`
- `multi-list.sh [OPTIONS] [N-SIZE] OUT-DIRECTORY IN-DIRECTORY`
- `unify.pl [OPTIONS] OUT-FILE {IN-FILE+|IN-DIRECTORY}`
- `split-unify.sh [OPTIONS] IN-DIRECTORY`

Note: items in square brackets are optional, in curly brackets either one or the other item can be supplied; + means one or more of these can be supplied.

Options in list.pl and multi-list.sh

-a --all_freq_combos	Produce figures for all possible frequency combinations (required if NSP's statistics.pl is to be used to calculate word association measures).
-f --frequency N	Suppresses the listing of n-grams that occur less than N times.
-h --help	Prints a help message.
-l --newline	Includes n-grams spanning across the new-line character. These are excluded by default.
-n --ngram N	Creates n-grams of N words each. N = 2 by default.
-o --stop FILE	Uses FILE as a stop list and removes n-grams containing at least one stop word (in ABSOLUTE mode) or only stop words (in ADDITIVE mode). Stop words should be declared as Perl Regular expressions in FILE.
-p --separator SEP	Uses SEP as the symbol(s) separating the words of an n-gram. By default, words in n-grams are separated by an interpunct (middle dot), but this can be changed using this option, for example to <> or _. Care needs to be taken that the separator symbol does NOT occur in the input texts, otherwise confusion will result.

-t --token FILE	Uses regular expressions in FILE as token definition. To display the default values, write *show* instead of FILE.
-v --verbose	Prints information as the programme runs.
-V --version	Prints the version number and copyright information.
-w --window N	Sets window size to N. By default, the window size is the same as the *n* of the n-grams; by specifying a larger window size, all combinations of *n* words (in the order in which they occur) inside the window will be included in the n-gram list (i.e. if 2-grams are extracted in a window size of 3, the n-grams produced will be of word 1 and word 2, word 2 and word 3, *as well as* word 1 and word 3).

Note: the long version of the options (starting with two hyphens) can only be used in `list.pl`.

Options exclusive to list.pl

-e --encoding ENC	Handles input and output files with the given character encoding. Default is utf8, but if necessary a different encoding can be forced using this option. A suitable token definition must also be supplied as the standard token definition is in utf8 (use -t option). Non-utf8 is not generally recommended and is here only provided for compatibility with legacy data sets. It is better to convert the data to utf8 as non-utf8 encodings have not been widely used in testing.
-s --set_freq_combo FILE	Uses the frequency combinations in FILE to decide which combinations of tokens to count in a given n-gram. By default, only the frequency of the complete n-gram is kept track of. This option is provided for compatibility with the NSP only; see the NSP's documentation for details.
-d --display_freq_combo	Prints out the frequency combinations used. If frequency combinations have been provided through –set_freq_combo above, these are output; otherwise the default combination is output.

Options exclusive to multi-list.sh

-d	Produces ONE output list per ONE input document.
-P ARG	Causes the output directory to have the ARG prepended to its name.
-S ARG	Causes the output directory to have the ARG appended to its name.
-H	Replaces hyphens (-) with 'HYPH' in output lists.

Options in unify.pl and split-unify.sh

-d --doc_count	Adds a count of how many documents (input files) each n-gram appears in. This figure is appended to the end of each line of output.
-h --help	Displays help.
-V --version	Displays version information.
-v --verbose	Displays processing information as the programme runs.

Options exclusive to unify.pl

-D --display_freq_combo	Shows the current frequency combinations setting.
-e --encoding ENC	Handles input and output files with the given character encoding. Default is utf8, but if necessary a different encoding can be forced using this option. Non-utf8 is not generally recommended and is here only provided for compatibility with legacy data sets. It is better to convert the data to utf8 as non-utf8 encodings have not been widely used in testing.
-s --set_freq_combo FILE	Uses the frequency combinations in FILE to decide which combinations of tokens to count in a given n-gram. If n-gram lists with more frequency combinations than the frequency of the n-gram are needed (such as when statistics of association should be calculated), this option MUST be used, together with the appropriate file. The lists to be combined must contain the necessary frequencies, of course.

Options exclusive to split-unify.sh

-i	Discards directory with individual lists (otherwise this directory is prefixed with `indiv_lists` and retained).
-m	Causes lists to be combined using an algorithm that minimises RAM requirements.
-n	Causes no numbers for calculation of statistical measures of association to be calculated even if present in the input files and no alphabet splitting is in effect.
-s	Uses processing mode 'small' by passing data directly to `unify.pl`.
-b	Runs big version for large amounts of data (84-way alphabet split).
-u	Leaves output lists untidy, that is, suppresses the insertion of tabs between the n-grams, their frequencies and any document counts.

Banerjee, S., & Pedersen, T. (2003). 'The Design, Implementation and Use of the Ngram Statistics Package'. In *Proceedings of the 4th International Conference on Intelligent Text Processing and Computational Linguistics*. Mexico City.

O'Donnell, M. B. (2011). 'The Adjusted Frequency List: A method to produce cluster-sensitive frequency lists'. *ICAME Journal*, 35 (April). Retrieved from: http://icame.uib.no/ij35/Matthew_Brook_ODonnell.pdf

Wilmsmann, B. (2007). *Re-write of Text-NSP.* (Original work published 2007). Retrieved from https://github.com/BjoernKW/Publications/blob/master/Re-write_of_Text-NSP.pdf