# LGram

*A cross–platform memory–efficient ngram calculator*

## Introduction

Ngrams are of the up most importance in computational and corpus linguistics. Lgram is a cross–platform tool for calculating ngrams in a memory–efficient manner. The current crop of n-gram tools have non–constant memory usage (see the Comparisons section) such that ngrams cannot be computed for large input texts. Given the prevalence of large texts in computational and corpus linguistics, this deficit is problematic. Lgram has constant memory usage so it can compute ngrams on arbitrarily sized input texts. Lgram achieves constant memory usages by periodically syncing the computed ngrams to an sqlite database stored on disk. The database has support for for around $256^8$ types each of which can occur $256^8$ times. Once all ngrams have been computed the database is used to create a sorted frequency list of ngrams (again with constant memory usage). The database can even be used to compute more exotic ngram forms (such as discontinuous ngrams) although this functionality is not yet provided by lgram.

The lgram processing pipeline proceeds as follows.

```
buffers -> blocks -> words -> ngrams
```

A buffer is a chuck of text; blocks are linguistic units as specified by the boundary characters (see the *-b* option). The blocks are split into words and the words are used to compute the ngrams. Special care is taken to make sure ngrams which transverse buffers are computed correctly. The advantage of this method is low memory usage.

## Usage

The functionality of lgram can be customised using command line options. Command–line options come in two equivalent forms; short (*e.g.* **-n**) and long (*e.g.* **--ngram**). Only the short form is available on windows. After all options have been specified a list of input files to process has to be provided (see Example section) unless **-f** is used. All specified input files are processed and stored in the same database.

### -n --ngram <m–n | n>

This option sets the order of the ngrams to be computed. The order can be specified as either a single number, *i.e.* -n 3 in which case the order is 1-3;

or the order can be specified as a range, *i.e.* 2-4. If the order is specified as 4-4 only ngrams of order 4 are computed. The maximum order is 9 and the default order is 1-4.

**-f --file-list <filename>**

Instead of reading input files from the command–line, read input files from *filename*. Each line of *filename* should be the name of a single file to process.

**-d --database <dbname>**

Lgram always creates a database when computing ngrams but by default that database is deleted when lgram finishes. Specifying the **-d** argument saves the database as *dbname*.

**-o --output-file <filename>**

By default, all ngrams are printed to stdout. Specifying the **-o** options also saves the ngrams to *filename*.

**-O --output-by-n <filename>**

Rather than saving the ngrams to a single file, option **-O** saves the ngrams to multiple files named *filename-k* where *k* is the order of the ngram (see Example section).

**-r --regex–file <filename>**

Apply each regex in *filename* to each block processed by lgrams. The regex file must be in the following format.

```
regex1   <tab>   replace_string1
# comment
regex2   <tab>   replace_string2
```

such that the regular expression and the replacement string are tab separated. The regex is a global replace and the regex format is PCRE (Perl compatible). If there is no replace string the regex matches are replaced with nothing (*i.e.* they are removed). For instance if the user wishes to remove all double quotes the following regex would suffice:

```
"\s*
```

without the `\s*` part, whatever succeeds the double quote may be ngram'ed with a space character.

**-b --boundary <str>**

Ngrams are not calculated across boundary characters. The default string is *.,?!:;* so that ngrams are not calculated across sentences or divisions of sentences.

**-B --ignore-boundary**

By default the boundary characters are treated as tokens and included in ngrams, if **-B** is enabled ngrams which contain boundary characters are ignored.

**-u --preserve-case**

By default all letters are transformed to lower case, **-u** prevents this behaviour.

**-s --no-stdout**

Do not print ngrams to stdout. This options should be used in conjunction with **-o**, **-O** and **-d**.

**-q --quiet**

Don't print anything but errors

## Example

Let us compute 1-grams to 4-grams on Bram Stokers Dracular[1]

```
lgram -n 4 -s -O BSD-ngram -d BSD.db bs-dracular.txt
```

Running this commands gives the output:

```
[1] bs-dracular.txt [29s]
[+] building 1gram frequency list
[+] building 2gram frequency list
[+] building 3gram frequency list
[+] building 4gram frequency list

processed 445537 ngrams in 41s
```

and the five files:

[1] `http://www.gutenberg.org/ebooks/345`

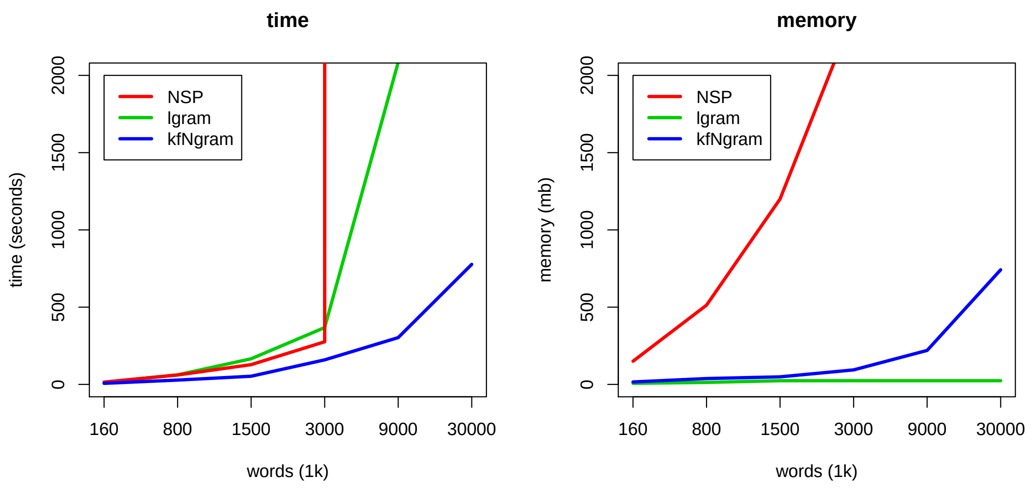
```
BSD.db
BSD-ngram-1
BSD-ngram-2
BSD-ngram-3
BSD-ngram-4
```

`BSD.db` is the sqlite database. If the **-d** argument was not specified, the database would have been deleted after the ngrams were saved. The `BSD-ngram` files contain sorted lists of ngrams and their frequency. For instance, the first 5 lines of `BSD-ngram-3` are:

```
i could see 71
i could not 65
there was a 63
i did not 56
a sort of 49
```

## Comparisons

For a comparisons we choose two widely used ngram tools; NSP[2] and kfNgram[3]. All tests were performed on a 2Ghz desktop computer, all settings were made as equivalent as possible. These values are for computing 4-grams across a range of literature.



We see that lgram has constant memory usage but is slower as a result. It takes around 7 hours to compute 4-grams on 30 million words. NSP is fast

[2]`http://www.d.umn.edu/~tpederse/nsp.html`

[3]`http://www.kwicfinder.com/kfNgram/kfNgramHelp.html`

but cannot calculate ngrams on texts above 3 million words on a computer with 4gb of ram. KfNgram performs best although it does not have constant memory usage and it cannot be run in Unix. In the future we would like to incorporate features of KfNgram to speed up lgram.

## Requirements

Lgram requires libpcre/libpcrecpp for regular expressions and libsqlite3 for the database. For windows the libraries are included in the precompiled executable of lgram but you must install the Microsoft Visual C++ 2010 Redistributable Package[4]. For Unix and Mac the libraries can be installed using your package manager or download from `pcre.org` and `sqlite.org`.

## Credits

Lgram was written by Edward J. L. Bell[5] at Lancaster University[6] and funded by UCREL[7]. The project was initiated by Dr Paul Rayson[8]. Lgram is open–source and released under the GPL license.

[4]`http://www.microsoft.com/download/en/details.aspx?id=14632`
[5]`ejlbell@gmail.com`
[6]`http://www.lancs.ac.uk`
[7]`http://ucrel.lancs.ac.uk`
[8]`http://www.comp.lancs.ac.uk/~paul`