Developer Overview of SynthaCorpus

Microsoft

July 26, 2017

The SynthaCorpus project is written in C11 and perl 5. Development has been carried out under cygwin on Windows 10, using gcc version 5.4.0 and perl 5.22. The C executables can also be built under Visual Studio 2015 and solution/project files are included in the distribution to facilitate this. Successful building and basic operation under Bash on Windows-10 and on Apple Macintosh has been confirmed though more thorough testing is needed. We don't expect major problems porting to other environments. However, because large data sizes lead to heavy memory demands, there is a requirement that the executables will be built for and run on a 64-bit architecture.

In the following it is assumed that the root of the cloned project directory structure is at SynthaCorpus. All references to directory paths are relative to that.

1 Using SynthaCorpus

Please see the companion User Overview document.

2 Licenses, Authorship and Copyright

Except for the third-party open-source code and content listed below the code in this project was written by Microsoft.

Project Gutenberg books - the directory ProjectGutenberg contains 49 popular books in text format, downloaded from http://www.gutenberg.org/. These are books which are no longer subject to copyright. They are provided as a small test set to illustrate and verify the operation of project SynthaCorpus.

Fowler Noll Vo hash function - this is a hashing function which we have consistently found to be fast and effective. The code is at src/imported/Fowler-Noll-Vo-hash. Both fnv.c and fnv.h in that directory include a licence allowing free distribution. See http://www.isthe.com/chongo/tech/comp/fnv/.

Tiny Mersenne Twister random number generator - this is a fast random number generator with very good properties. We chose to use the tiny version (still with excellent properties), to allow the possibility of corpus generation being built into an indexer being studied. The code is at src/imported/TinyMT_cutdown. The file readme.html contains the license. See http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index.html

3 Simple Text ARChive (STARC) format

As noted in the User Overview, STARC is the recommended format for use with SynthaCorpus. A STARC is a UTF-8 text file consisting of a concatenation of records. Three types of records are defined: Document (D), Header (H), and Trailer (T). D records are the most important. H and T records are entirely optional but can be used to contain metadata about a D record. SynthaCorpus extracts properties from D records only. When generating, it puts a document identifier into an H record preceding the D record containing the generated text.

A STARC record starts with a space, then a decimal length L represented in ASCII digits, then a single character specifying the record type (H, D, or T), another space, followed by L bytes of the actual document content. This form of representation allows arbitrary content including control characters and NULs.

Here is an example STARC containing four records. The dollar signs are not part of the archive; they are included to unambigously mark the beginning and end of the STARC.
r represents RETURN and
n represents LINEFEED.

```
\$ 4H Doc1 19D A quick brown fox\r\n 4H Doc2 22D Jumped over the lazy\r\n$
```

A wide variety of other formats have been used in publicly distributed corpora such as those of TREC and CLEF. Perhaps in future `SynthaCorpus` will be extended to handle them directly. In the meantime, you would need to develop a converter from whatever other format into STARC format. To assist in this process, you may wish to look at `convertPGtoSTARC.c`, the provided converter from Project Gutenberg format into STARC. Tools for checking the validity of a STARC file and for counting documents are also provided: `checkSTARCFile.exe` and `countDocsInCorpus.exe`

4 Project dependencies

Successful building and operation of the `SynthaCorpus` project requires installation of the following packages. Version numbers shown are the ones used. Other versions may be suitable.

- `gcc 5.4.0` and `make 4.2.1` OR VisualStudio 2015
- `perl 5.22`
- `gnuplot 5.0` – for displaying extracted corpus properties
- `pdflatex 2.9.4902` – for compiling documentation (if needed)
- `acroread 2017 version` OR other PDF viewer – for viewing documentation and plots.

5 Building executables and documents

5.1 With gcc

As noted in the User Overview, you can build the gcc executables by running the Exerciser script. If you want to do it manually,

```
cd src
make cleanest
make
```

builds all the executables. One warning is emitted, complaining about a function which is not referenced but which may be in future. Currently, the executables remain in `src`.

5.2 With VS 2015

The relevant VS solution file is in `src/BuildAllExes`. It includes the projects which build eight executables in the `src/BuildAllExes` directory.

Properties are only set for the `Release`, `x64` combination. Changes to defaults for this combination include:

- Project/General: Use multi-byte characters
- C/C++/Preprocessor: Define symbols `WIN64` and `_CRT_SECURE_NO_WARNINGS`

- C/C++/Precompiled headers: Not using precompiled headers
- C/C++/Advanced: Compile as C

Open the solution file with VS 2015 and click Build/Rebuild.

5.3 Testing the system once built

The exerciser script runs each of the main components and can be used as a basic sanity test. For a more thorough test add the '`thorough`' option to the `exerciser.pl` command line.

5.4 Rebuilding PDF documents

In the uncommon case that you want to rebuild the documentation PDFs, you will need to have a TeX distribution installed plus a few additional packages, easily obtained via CTAN.

```
cd doc
rm *.pdf
pdflatex user_overview
pdflatex developer_overview
pdflatex how_to_synthesize_a_corpus
```

6 Notes on coding

Project `SynthaCorpus` has been developed in the course of scientific research rather than engineered to commercial standards. In many cases, but not always, there is quite extensive internal documentation at the head of modules or critical functions. Unfortunately, there is inconsistency in variable naming. The process of moving from the use_of_underscores to "camelCase" is regrettably incomplete.

Be warned that the principal author is highly enamoured of the `unless` clause in perl. Most scripts contain many examples of:

```
die "message"
   unless <condition>;
```

The C programs make very extensive use of `dahash` hash tables, defined in `src/utils/dahash.[ch]`. These hash tables avoid the use of buckets. Their size is constrained to be a power of two and collisions are handled by relatively prime rehash. Once a table reaches a specified percentage of capacity, the table size is doubled behind the scenes. The hash function is the Fowler Noll Vo one.

Example code for using the `dahash` system is in `src/utils/dahashDemo.c`.

SynthaCorpus also defines a mechanism for dynamic arrays – `src/utils/dynamicArrays.[ch]`. Accessing an array element beyond allocated storage, causes the array to grow using a configurable strategy.

C programs make extensive use of memory mapping through wrappers in `src/utils/general.[ch]` which use either the Windows style `CreateFileMapping()/MapViewOfFile()` (used if the WIN64 symbol is defined) or the Unix interface `mmap()`.

All programs assume the use of UTF-8. We recommend the use of the iconv library `https://www.gnu.org/software/libiconv/` for converting other character sets into UTF-8. Functions for dealing with UTF-8 are in `src/characterSetHandling/unicode.[ch]`. Also in that directory is a perl script for updating Unicode tables from the database at `http://unicode.org/`.

7 Layout of Project Directories

The project root directory contains the following files and directories:

`doc` - Contains the LaTeX source and PDFs of project documentation.

`Experiments` - This is the directory under which all the output produced by scripts and executables will be stored. If it doesn't exist the main perl scripts will create it.

`LICENCE.TXT` - See beginning of this document.

`ProjectGutenberg` - Text versions of 49 books used to illustrate and validate the executables in this project.

`src` - All the perl and C sources. Currently executables are built in or under this directory.

8 Desirable future developments

1. Code tidyup - variable names, internal documentation, module structure. [Boring but useful]
2. Additional document formats - or maybe converters from other formats into STARC format? [Boring but useful]
3. Piecewise growth models - Currently scaling up from a small corpus assumes the use of only a simple linear model for word frequency distribution. [A challenge and very useful contribution.]
4. Extending the supported term dependency models to include term repetition within a document, and term co-occurrence as well as n-grams. [A challenge and very useful contribution.]
5. Efficiency monitoring and improvement, particularly reducing memory demands without exploding runtime. [Could increase scope of application.]
6. Reduce memory demands of `corpusGenerator.exe` by using more memory-efficient representations of Markov transition matrices. [Could increase scope of application.]